PEMROGRAMAN PARALEL BERORIENTASI OBJEK PADA RUTIN KOMUNIKASI KOLEKTIF MENGGUNAKAN STANDAR MPI1

TUGAS AKHIR

Diajukan Sebagai Salah Satu Syarat
untuk Memperoleh Gelar Sarjana Teknik pada
Jurusan Teknik Elektro

UIN SUSKA RIAU

Oleh :

SITI HABIBAH

10655004562

FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SULTAN SYARIF KASIM RIAU
PEKANBARU

2013

PEMROGRAMAN PARALEL BERORIENTASI OBJEK PADA RUTIN KOMUNIKASI KOLEKTIF MENGGUNAKAN STANDAR MPI1

SITI HABIBAH
NIM: 10655004562

Tanggal Sidang: 27 Juni 2013
Tanggal Wisuda: Nopember 2013

Jurusan Teknik Elektro
Fakultas Sains dan Teknologi
Universitas Islam Negeri Sultan Syarif Kasim Riau
Jl. Soebrantas No. 155 Pekanbaru

ABSTRAK

Pada MPI1 yang mendukung komunikasi kolektif, sampai saat ini masih mendukung pemrograman modular. Oleh karena itu pada penelitian ini dikembangkan kode program paralel yang berorientasi objek pada rutin komunikasi kolektif. Tujuan penelitian ini yaitu mengembangkan rutin komunikasi kolektif berorientasi objek pada pemrograman paralel yang menggunakan standar MPI1. Pada penelitian ini jaringan fisik dibangun dengan sistem operasi *Linux Pelicanhpc-v2.2* berbasis *Debian.* Dan telah berhasil dibangun satu *head node* dan dua *compute node* yang dapat menjalankan kompilasi kode program paralel dan menganalisis perbandingan waktu eksekusi antara kode paralel C dengan kode paralel C++.

Kata Kunci : *Compute node, head node,* komunikasi kolektif, MPI1, program paralel

OBJECT ORIENTED PROGRAMMING PARALLEL ROUTINE ON COLLECTIVE COMMUNICATION USING STANDARD MPI1

SITI HABIBAH
NIM : 10655004562

Date of Final Exam : June 27th 2013
Date of Graduation Ceremony : November, 2013

Departement of Electrical Engineering
Faculty of Science and Technology
State Islamic University of Sultan Syarif Kasim Riau
Soebrantas Street No. 155 Pekanbaru

ABSTRACT

On MPI1 that support collective communication, it is still supported by modular programming. Therefore, in this research, it is developed parallel program codes based on object oriented in collective communication routines. The purpose of this research is to develope an object oriented collective communication routines on parallel programming using MPI1 standard. In this research, the physical network is built with the Linux Debian operating system of Pelicanhpc-v2.2. It has succesfully built one head node and two compute nodes that can run the compilation of parallel code and analyze the comparison of execution time between the parallel C code and the parallel C++ code.

Keywords: Communication collective, compute node, head node, MPI1, parallel programming

KATA PENGANTAR

Assalammu'alaikum wa rahmatullahi wa barakatuh.

Alhamdulillah hirabbil'alamin, puji syukur atas segala rahmat dan kekuatan yang diberikan Allah SWT, penulis dapat menyelesaikan tugas akhir dengan judul **"**Pemrograman Paralel Berorientasi Objek Pada Rutin Komunikasi Kolektif Menggunakan Standar Mpi1".

Pada kesempatan ini penulis, mengucapkan terima kasih kepada semua pihak yang membantu penulis baik itu berupa moral, materil, ataupun berupa pikiran sehingga terlaksananya penelitian dan penulisan laporan ini terutama sekali kepada :

1. Bapak Prof. DR. H.M. Nazir, selaku Rektor Universitas Islam Negeri Sultan Syarif Kasim Riau.
2. Ibu Dra. Hj. Yenita Morena, M.Si selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Sultan Syarif Kasim Riau.
3. Bapak Kunaifi, ST, PgDipEnST, M.Sc selaku Ketua Jurusan Teknik Elektro Fakultas Sains dan Teknologi Universitas Islam Negeri Sultan Syarif Kasim Riau.
4. Ibu Zulfatri Aini, ST., MT selaku Sekretaris jurusan Teknik Elektro Fakultas Sains dan Teknologi Universitas Islam Negeri Sultan Syarif Kasim Riau.
5. Bapak Edmond Febrinicko Armay, S.Si., MT, selaku dosen pembimbing yang senantiasa memberikan arahan-arahan dan masukan-masukan yang sangat membantu penulis dalam menyelesaikan tugas akhir ini.
6. Bapak Dr. Alex Wenda, ST.,M.Eng dan Bapak Sutoyo, ST.,MT, selaku dosen penguji I dan dosen penguji II yang banyak memberikan pertanyaan, masukan dan saran demi sempurnanya tugas akhir ini.
7. Ibu Dian Mursyitah, ST., Bapak Suwanto Sanjaya, ST., dan Ibu Ewi Ismaredah, S.Kom., M.Kom yang membimbing dan menguji penulis.

8. Seluruh Dosen Jurusan Teknik Elektro Universitas Islam Negeri Sultan Syarif Kasim Riau yang telah memberikan ilmu dan pengetahuan yang bermanfaat kepada penulis selama mengikuti perkuliahan di Jurusan Teknik Elektro.
9. Kepada Ibu dan ayah (alm) tercinta, yang sangat penulis sayangi atas segala do'a, nasihat dan kasih sayangnya yang tiada terhingga besarnya.
10. Seluruh keluarga besar penulis atas perhatian yang telah diberikan, terkhusus buat Syafriadi dan Syurbayani.
11. Kepada sahabat-sahabat serta teman seperjuangan TE '06, terutama untuk cewek-cewek TE '06 :Ana, Wike, Athul, Dian, Ades, Ai, Mardha, Rofa yang senantiasa memberikan dukungan dan semangat untuk terus berjuang.
12. Rekan-rekan Teknik Elektro angkatan 2007, 2008 Universitas Islam Negeri Sultan Syarif Kasim Riau, yang senasib dan seperjuangan dalam memperoleh kelulusan.
13. Seseorang yang terspesial yang rela berkorban bagi penulis dalam membantu menyelesaikan Tugas Akhir ini.
14. Seluruh pihak yang ikut membantu terselesaikannya tugas akhir ini yang tidak dapat penulis sebutkan satu per satu.

Penulis menyadari sepenuhnya bahwa tugas akhir ini jauh dari kesempurnaan. Kritik dan saran sangat penulis harapkan jika terdapat kekurangan. Penulis berharap semoga penelitian ini bernilai karya yang dapat memberikan sumbangan bagi kemajuan ilmu pengetahuan dan bermanfaat bagi pembacanya. Amin.

Pekanbaru, 27 Juni 2013

Penulis

DAFTAR ISI

Halaman

LEMBAR PERSETUJUAN ..... **ii**

LEMBAR PENGESAHAN ..... **iii**

LEMBAR HAK ATAS KEKAYAAN INTELEKTUAL ..... **iv**

LEMBAR PERNYATAAN ..... **v**

LEMBAR PERSEMBAHAN ..... **vi**

ABSTRAK ..... **vii**

ABSTRACT ..... **viii**

KATA PENGANTAR ..... **ix**

DAFTAR ISI ..... **xi**

DAFTAR GAMBAR ..... **xiii**

DAFTAR TABEL ..... **xiv**

DAFTAR SINGKATAN ..... **xvi**

DAFTAR LAMPIRAN ..... **xvii**

BAB I PENDAHULUAN

1.1 Latar Belakang I-1

1.2 Rumusan Masalah I-3

1.3 Batasan Masalah I-3

1.4 Tujuan Penelitian I-4

1.5 Sistematika Penulisan I-4

BAB II LANDASAN TEORI

2.1 Pemrosesan Paralel II-1

2.2 *Message Passing Interface* (MPI) II-1

2.3 Pemrograman Berorientasi Objek II-2

2.4 Bahasa C++ II-3

2.5 Rutin Komunikasi Kolektif II-6

BAB III METODOLOGI PENELITIAN DAN PERANCANGAN DIAGRAM ALIR (*FLOWCHART*)

3.1 Metodologi Penelitian III-1

3.2 Diagram Alir (*Flowchart*) III-2

3.2.1 *Flowchart* program Sinkronisasi *Barrier* III-3

3.2.2 *Flowchart* program *Broadcast* III-5

3.2.3 *Flowchart* program *Gather* dan *Scatter* III-7

3.2.4 *Flowchart* program *Gather-to-All* III-11

3.2.5 *Flowchart* program *All-to-All Scatter* dan *Gather* III-13

3.2.6 *Flowchart* program *Reduce* III-16

3.2.7 *Flowchart* program *Reduce-Scatter* III-20

3.2.8 *Flowchart* program *Scan* III-23

BAB IV ANALISIS KODE PROGRAM

4.1 Kode Program Sinkronisasi *Barrier* IV-1

4.2 Kode Program *Broadcast* IV-2

4.3 Kode Program *Gather* dan *Scatter* IV-3

4.4 Kode Program *Gather-to-All* IV-4

4.5 Kode Program *All-to-All Scatter* dan *Gather* IV-5

4.6 Kode Program *Reduce* IV-7

4.7 Kode Program *Reduce-Scatter* IV-8

4.8 Kode Program *Scan* IV-9

BAB V PENUTUP

5.1 Kesimpulan V-1

5.2 Saran V-2

DAFTAR PUSTAKA

LAMPIRAN

DAFTAR RIWAYAT HIDUP

BAB I
PENDAHULUAN

1.1 Latar Belakang

Seiring dengan perkembangan komputer kebutuhan akan komputasi juga semakin kompleks. Hampir tidak ada segi kehidupan manusia di mana komputer tidak terlibat di dalamnya. Bahkan beberapa kasus tidak dapat diselesaikan dengan pemrograman konvensional, yaitu pemrograman berparadigma sekuensial. Oleh karena itu, dengan melibatkan semua kemampuan prosesor yang lebih dari satu diharapkan dapat menyelesaikan permasalahan tersebut menggunakan pemrograman berbasis paralel (Kurniawan, 2010).

Pemrograman paralel adalah teknik pemrograman komputer yang memungkinkan eksekusi perintah/operasi secara bersamaan, baik dalam komputer dengan satu CPU (prosesor tunggal) maupun dengan banyak CPU (prosesor ganda dengan mesin paralel). Semakin banyak hal yang bisa dilakukan secara bersamaan (dalam waktu yang sama), semakin banyak pekerjaan yang bisa diselesaikan. Analogi yang paling mudah adalah bila merebus air sambil memotong-motong bawang saat memasak, waktu yang dibutuhkan akan lebih sedikit dibandingkan bila mengerjakan hal tersebut secara berurutan (serial). Atau waktu yang dibutuhkan memotong bawang akan lebih sedikit jika dikerjakan berdua (Prasetyo, 2011).

Namun komunikasi dan sinkronisasi diantara unit pemroses (*processing unit*) menjadi salah satu diantara tantangan terbesar untuk menghasilkan kode program paralel dengan performa yang baik.

Pada pemrograman paralel bahasa pemrograman yang umum digunakan adalah bahasa C dan C++. Tetapi tidak diketahui apakah bahasa C atau bahasa C++ dapat mempercepat kinerja pemrograman paralel atau malah memperlambatnya (Ajinagoro, 2005).

Bahasa C merupakan bahasa pemrograman terstruktur, sedangkan bahasa C++ merupakan bahasa pemrograman berorientasi objek.

Pada penelitian ini dilakukan perbandingan waktu eksekusi antara pemrograman paralel terstruktur dan pemrograman paralel berorientasi objek atau antara bahasa C dan bahasa C++, agar dapat membantu *programmer* dalam memilih bahasa pemrograman yang tepat dan memiliki kecepatan waktu eksekusi yang lebih cepat.

Untuk menghasilkan kode program paralel dengan performa yang baik dibutuhkan pemrograman dengan paradigma *message-passing interface* atau disingkat dengan MPI. MPI merupakan sebuah protokol komunikasi yang sifatnya *language independent*, *portable* dalam men-*support* berbagai *platform*, dan memiliki spesifikasi semantik yang mengatur bagaimana perilaku setiap implementasinya. MPI mendukung komunikasi baik dengan tipe *point-to-point* maupun yang bersifat kolektif. Komunikasi kolektif merupakan sebuah komunikasi yang melibatkan sebuah *group* atau kumpulan *group* dari proses. Proses pertukaran pesan secara sederhana pada komunikasi kolektif dapat menggunakan *MPI_Bcast* dan *MPI_Reduce* (Hasan, 2011). MPI menjamin *message* yang dibuat oleh komunikasi kolektif tidak akan tercampur dengan *message* yang dibuat pada komunikasi *point-to-point*. MPI terdiri dari dua kategori yakni MPI1 dan MPI2. Standar MPI1 yang mendukung komunikasi kolektif antara lain (Kurniawan, 2010):

1. Sinkronisasi *barrier*
2. *Broadcast*
3. *Gather* dan *Scatter*
4. *Gather-to-All*
5. *All-to-All Scatter* dan *Gather*
6. *Reduce*
7. *Reduce-Scatter*
8. *Scan*

Pada MPI1 yang mendukung komunikasi kolektif, sampai saat ini masih mendukung pemrograman modular. Oleh karena itu pada penelitian ini dikembangkan kode program paralel yang berorientasi objek pada rutin komunikasi kolektif yang berstandarkan MPI1.

Dengan diterapkannya pemrograman paralel berorientasi objek diharapkan nantinya dapat meningkatkan kinerja komputasi khususnya dalam menyelesaikan beban komputasi yang tinggi.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, dapat dirumuskan suatu permasalahan yaitu bagaimana membandingkan pemrograman paralel yang berorientasi objek pada rutin komunikasi kolektif menggunakan standar MPI1 dan melakukan analisis dengan melakukan perbandingan waktu eksekusi antara pemrograman paralel terstruktur dengan pemrograman paralel berorientasi objek.

1.3 Batasan Masalah

Dalam penelitian Tugas Akhir ini masalah dibatasi sebagai berikut:

1. Bahasa pemrograman yang digunakan yaitu bahasa C dan bahasa C++.
2. Pustaka pemrograman yang digunakan adalah standar MPI1.
3. Rutin yang diteliti yaitu rutin komunikasi kolektif antara lain:
    1. Sinkronisasi *barrier*
    2. *Broadcast*
    3. *Gather* dan *Scatter*
    4. *Gather-to-All*
    5. *All-to-All Scatter* dan *Gather*
    6. *Reduce*
    7. *Reduce-Scatter*
    8. *Scan*
4. Membandingkan waktu eksekusi antara kode paralel C dan kode paralel C++.

1.4 Tujuan Penelitian

Adapun tujuan dari penelitian tugas akhir ini yaitu membandingkan rutin komunikasi kolektif berorientasi objek pada pemrograman paralel yang menggunakan standar MPI1 dan menganalisis perbandingan waktu eksekusi antara pemrograman paralel terstruktur dengan pemrograman paralel berorientasi objek pada rutin komunikasi kolektif serta memperkaya literatur dalam bidang pemrosesan paralel.

1.5 Sistematika Penulisan

Sistematika penulisan laporan ini dibagi menjadi lima bab, hal ini dimaksudkan agar dalam penulisan laporan Tugas Akhir dapat diketahui tahapan dan batasannya. Adapun sistematika penulisannya adalah sebagai berikut :

BAB I PENDAHULUAN

Pada bab ini menguraikan secara umum dan singkat mengenai latar belakang, rumusan masalah, batasan masalah, tujuan penelitian dan sistematika penulisan.

BAB II LANDASAN TEORI

Pada bab ini berisi mengenai teori pendukung dari analisis yang diuji.

BAB III METODOLOGI DAN PERANCANGAN DIAGRAM ALIR

Pada bab ini berisi mengenai metodologi penelitian, analisis kebutuhan jaringan, dan perancangan diagram alir (*flowchart*).

BAB IV ANALISIS KODE PROGRAM

Pada bab ini berisi mengenai perbandingan waktu eksekusi antara pemrograman paralel terstruktur dengan pemrograman paralel berorientasi objek pada rutin komunikasi kolektif.

BAB V KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan yang diperoleh dari penelitian pada bab sebelumnya dan saran-saran dari pengamatan

BAB II
LANDASAN TEORI

2.1 Pemrosesan Paralel

Pemrosesan paralel adalah suatu operasi komputer dengan dua program atau lebih yang dikerjakan secara serentak atau bersama-sama dengan menggunakan beberapa komputer *independent* (Liswandini, 2010). Ini umumnya diperlukan saat kapasitas yang dibutuhkan sangat besar, baik karena harus mengolah data dalam jumlah besar ataupun karena tuntutan proses komputasi yang banyak (Prasetyo, 2011).

Pemrosesan paralel dapat mempersingkat waktu eksekusi suatu program dengan cara membagi suatu program menjadi bagian-bagian yang lebih kecil yang dapat dikerjakan pada masing-masing prosesor secara bersamaan. Tujuan utama dari pemrosesan paralel adalah untuk meningkatkan performa komputasi. Semakin banyak hal yang bisa dilakukan secara bersamaan (dalam waktu yang sama), semakin banyak pekerjaan yang bisa diselesaikan (Prasetyo, 2011).

Untuk melakukan aneka jenis komputasi paralel ini diperlukan infrastruktur mesin paralel yang terdiri dari banyak komputer yang dihubungkan dengan jaringan dan mampu bekerja secara paralel untuk menyelesaikan satu masalah. Untuk itu diperlukan aneka perangkat lunak pendukung yang biasa disebut sebagai *middleware* yang berperan untuk mengatur distribusi pekerjaan antar *node* dalam satu mesin paralel. Selanjutnya pemakai harus membuat pemrograman paralel untuk merealisasikan komputasi (Prasetyo, 2011).

2.2 *Message Passing Interface* (MPI)

MPI menyediakan standar pemakaian secara luas untuk menulis program pertukaran pesan. Kegunaan MPI yang lain yaitu (Barney, 2010):

1. Menulis kode paralel secara *portable*
2. Mendapatkan performa yang tinggi dalam pemrograman paralel
3. Menghadapi permasalahan yang melibatkan hubungan data *irregular* atau dinamis yang tidak begitu cocok dengan model data paralel

Hampir semua fungsi-fungsi atau *routine* menggunakan komunikator sebagai argumen. Salah satu komunikator yang paling sering digunakan adalah MPI_COMM_WORLD (Barney, 2010).

Di dalam program MPI, harus terdapat pemanggilan fungsi *MPI_Init* sebagai inisialisasi program dan *MPI_Finalize* untuk terminasi program. Deklarasi variabel dan algoritma_algoritma yang berada diantara *MPI_Init* dan *MPI_Finalize* akan dijalankan secara paralel atau yang dibaca oleh semua proses yang berada di dalam komunikator (Barney, 2010).

2.3 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*Object Oriented Programming)* disingkat OOP merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Teknik pemrograman dapat mencakup fitur seperti data abstraksi, enkapsulasi, modularitas, *polymorphisme*, dan pewarisan. Banyak bahasa pemrograman modern sekarang mendukung pemrograman berorientasi objek. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya (Pratami, 2010).

Pemrograman berorientasi objek sebenarnya telah berkembang sejak tahun 1962 dengan berpatokan bahwa objek adalah simulasi, sedangkan tekniknya menggunakan *classes, inheritance, polymorphisme,* dan *virtual functions*. Pemrograman ini berorientasi yang membungkus data dan fungsi dalam bentuk *class* dan objek, dimana mereka dapat saling berkirim, menerima, dan memproses pesan/data antar objek (Pratami, 2010).

Pemrograman berorientasi objek kini sudah sangat popular dalam dunia rekayasa perangkat lunak. Dengan menggunakan 3 konsep utama, *Encapsulate* (Enkapsulasi), *Inheritance* (Penurunan), dan *Polymorphic*. Ketiga konsep ini menjadi landasan kuat pada pemograman berorientasi objek yang dapat meningkatkan *readability* (mudah dibaca dan dipelajari), *reusability* (mudah digunakan kembali), dan *dekomposisi* permasalahan (pemodulan). Objek menjadi kunci utama keberhasilan OOP karena mengurangi tingkat kompleksitas dalam

pemrograman dan memungkinkan *maintenance* yang relatif lebih mudah terhadap perangkat lunak itu sendiri (Pratami, 2010).

Pemrograman berorientasi objek menekankan konsep berikut (Pratami, 2010):

1. Enkapsulasi

   Enkapsulasi adalah suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem sekaligus menyederhanakan penggunaan sistem itu sendiri. Akses ke *internal* sistem diatur sedemikian rupa melalui seperangkat *interface*.

2. Pewarisan (*Inheritance*)

   Sebenarnya manusia terbiasa melihat objek yang berada di sekitarnya tersusun secara hierarki berdasarkan *class*-nya masing-masing. Dari sini kemudian timbul suatu konsep tentang pewarisan yang merupakan suatu proses dimana suatu *class* diturunkan dari *class* lainnya sehingga ia mendapatkkan ciri atau sifat dari *class* tersebut.

3. *Polymorphism*

   Polymorphism berasal dari bahasa Yunani yang berarti banyak bentuk. Dalam Pemrograman Berorientasi Objek, konsep ini memungkinkan digunakannya suatu *interface* yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Dalam konsep yang lebih umum sering kali *polymorphism* disebut dalam istilah satu *interface* banyak aksi.

2.4 Bahasa C++

Tahun 1978, Brian W. Kerninghan dan Dennis M. Ritchie dari AT & T *Laboratories* mengembangkan bahasa B menjadi bahasa C. Bahasa B yang diciptakan oleh Ken Thompson sebenarnya merupakan pengembangan dari bahasa BCPL yang diciptakan oleh Martin Richard. Sejak tahun 1980, bahasa C banyak digunakan pemrogram di Eropa yang sebelumnya menggunakan bahasa B dan BCPL. Dalam perkembangannya, bahasa C menjadi bahasa paling populer diantara bahasa lainnya, seperti PASCAL, BASIC, FORTRAN (Ayuliana, 2004).

Tahun 1989, dunia pemrograman C mengalami peristiwa penting dengan dikeluarkannya standar bahasa C oleh ANSI. Bahasa C yang diciptakan Kerninghan dan Ritchie kemudian dikenal dengan nama ANSI C. Mulai awal tahun 1980, Bjarne Stroustrup dari AT & T Bell *Laboratories* mulai mengembangkan bahasa C. Pada tahun 1985, lahirlah secara resmi bahasa baru hasil pengembangan C yang dikenal dengan nama C++. Sebenarnya bahasa C++ mengalami dua tahap evolusi. C++ yang pertama, dirilis oleh AT & T *Laboratories*, dinamakan *cfront.* C++ versi kuno ini hanya berupa kompiler yang menerjemahkan C++ menjadi bahasa C (Ayuliana, 2004).

Pada evolusi selanjutnya, Borland International Inc. mengembangkan kompiler yang mampu mengubah C++ langsung menjadi bahasa mesin (*assembly*). Sejak evolusi ini, mulai tahun 1990 C++ menjadi bahasa berorientasi objek yang digunakan oleh sebagian besar pemrogram profesional (Ayuliana, 2004).

Pada subbab ini dijelaskan suatu contoh kerangka program seperti terlihat pada Gambar 2.1 sebagai berikut:

```
//my program
#prepocessor directive
main()
{
 // Batang Tubuh Program Utama
}
```

Gambar 2.1. Contoh kerangka program

Penjelasan kerangka program di atas yaitu:

1. Komentar

   Semua baris yang diawali dengan dua tanda *slash* (//) dianggap komentar dan tidak memiliki efek apapun pada perilaku program. *Programmer* dapat menggunakannya untuk memasukkan penjelasan singkat atau observasi dalam kode sumber.

2. *Preprocessor directive*

   Baris yang diawali dengan tanda *hash (#)* adalah direktif untuk preprosesor misalnya *#include, #define* dan lain sebagainya. Dalam penelitian ini yang digunakan yaitu *#include.* Dalam bahasa C, untuk melakukan *input output*

digunakan *library standard* bernama *stdio.h* sedangkan di dalam bahasa C++ digunakan c*stdio*. Preprosesor memiliki kemampuan menambahkan dan menghapus kode dari sumber. Preprosesor selalu dijalankan terlebih dahulu pada saat proses kompilasi terjadi. Kode ini tidak diakhiri dengan tanda *semicolon* (;), karena bentuk tersebut bukanlah suatu bentuk pernyataan tetapi merupakan *preprocessor directive* (Harahap, 2002).

3. Fungsi *main()*

   Baris ini berhubungan dengan awal definisi fungsi utama. Kata *main* diikuti oleh sepasang tanda kurung (). Sesudah sepasang tanda kurung, tubuh fungsi *main* dilampirkan dalam tanda *braces* ({}). Fungsi ini menjadi awal dan akhir eksekusi program C++, *main* adalah nama judul fungsi. Melihat bentuk seperti itu dapat diambil kesimpulan bahwa batang tubuh program utama berada didalam fungsi *main()*. Berarti dalam setiap pembuatan program utama, dapat dipastikan seorang pemrogram menggunakan minimal sebuah fungsi (Pradana, 2010).

4. Kurung kurawal

   Kurung kurawal buka menandakan awal program. Sedangkan kurung kurawal tutup menandakan akhir program.

2.5 Rutin Komunikasi Kolektif

Komunikasi kolektif merupakan sebuah komunikasi yang melibatkan sebuah *group* atau kumpulan *group* dalam ruang lingkup komunikator. MPI menjamin *message* yang dibuat oleh komunikasi kolektif tidak akan tercampur dengan *message* yang dibuat pada komunikasi *point-to-point*. Semua proses yang *default* (gagal), adalah anggota dalam komunikator *MPI_COMM_WORLD*, dan tanggung jawab *programmer* untuk memastikan bahwa semua proses pada komunikator berpartisipasi dalam operasi kolektif (Barney, 2011).

MPI menyediakan fungsi komunikasi data kolektif di mana semua proses terlibat dalam proses komunikasi data (Jamal, 2006). Standar MPI1 yang mendukung komunikasi kolektif antara lain (Kurniawan, 2010):

1. Sinkronisasi *barrier*
2. *Broadcast*

3. *Gather* dan *Scatter*
4. *Gather-to-All*
5. *All-to-All Scatter* dan *Gather*
6. *Reduce*
7. *Reduce-Scatter*
8. *Scan*

Pada MPI1 yang mendukung komunikasi kolektif, sampai saat ini masih mendukung pemrograman modular. Oleh karena itu pada penelitian ini dikembangkan kode program paralel yang berorientasi objek pada rutin komunikasi kolektif yang berstandarkan MPI1.

Semua proses pada komunikasi kolektif terdapat dalam komunikator panggilan rutinitas yang sama. Aplikasi *portable* harus mengasumsikan bahwa rutinitas kolektif termasuk sinkronisasi global (Anonymous, 2006). Pembatasan dan pertimbangan pemrograman pada komunikasi kolektif antara lain (Barney, 2011):

1. Operasi kolektif adalah *blocking*
2. Rutin komunikasi kolektif tidak mengambil *argument tag message*
3. Operasi kolektif dalam subset proses dapat dilakukan dengan partisi pertama subset menjadi *group-group* baru dan kemudian melampirkan *group* baru untuk komunikator baru
4. Komunikasi kolektif hanya dapat digunakan dengan tipe data standar MPI tidak dengan MPI *derived data types*

Fungsi kolektif melibatkan komunikasi diantara semua proses dalam kelompok proses. Operasi yang melakukan tugas-tugas yang canggih dideklarasikan pada rutin komunikasi kolektif di bawah ini antara lain (Forum MPI, 2009):

1. MPI_Barrier

Membuat sinkronisasi *barrier* dalam kelompok. *MPI_Barrier* melakukan *blocking* semua proses yang terjadi sampai semua proses pada *group comm* sudah mencapai *MPI_Barrier*.

MPI_Barrier(comm)

2. *MPI_Bcast*

   Broadcast (menyiarkan) pesan dari proses dengan *rank "root"* ke seluruh proses lain dalam *group*.

   MPI_Bcast(&buffer,count,datatype,root,comm)

3. *MPI_Scatter* dan *Gather*

   MPI_Scatter

   Mengirim data dari satu *task* ke seluruh *task* lain dalam grup.

   MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root, comm)

   MPI_Gather

   Mengumpulkan pesan berbeda dari setiap tugas dalam *group* untuk tujuan tugas tunggal. Rutin ini adalah operasi kebalikan dari *MPI_Scatter*.

   MPI_Gather(&sendbuf,sendcnt,sendtype,&recvbuf,recvcount,recvtype,root, comm)

4. *MPI_Allgather*

   Mengumpulkan data dari seluruh *task* dan mendistribusikannya untuk semua. Setiap tugas dalam *group*, pada dasarnya melakukan satu ke semua operasi penyiaran dalam *group*.

   MPI_Allgather(&sendbuf,sendcount,sendtype,&recvbuf,recvcount,recvcount,recvtype,comm)

5. *MPI_Reduce*

   Mengurangi nilai seluruh proses ke nilai tunggal.

   MPI_Reduce(&sendbuf,&recvbuf,count,datatype,op,root,comm)

6. *MPI_Reduce_scatter*

   Menggabungkan nilai dan menyebarkan hasilnya. Pertama melakukan pengurangan elemen bijaksana pada *vector* di semua tugas dalam *group*. Selanjutnya, hasil *vector* dibagi menjadi segmen beririsan dan didistribusikan di seluruh tugas. Hal ini setara dengan *MPI_Reduce* diikuti dengan operasi *MPI_Scatter*.

   MPI_Reduce_scatter(&sendbuf,&recvbuf,recvcount,datatype,op, comm)

7. *MPI_Alltoall*

   Mengirim data dari seluruh proses ke seluruh proses.

   MPI_Alltoall(&sendbuf,sendcount,sendtype,&recvbuf,recvcnt,recvtype,comm)

8. *MPI_Scan*

   Menghitung pindaian data koleksi proses.

   MPI_Scan(&sendbuf,&recvbuf,count,datatype,op,comm)

Fungsi *interface* yang sederhana yaitu *MPI_Bcast (broadcast)* dan *MPI_Reduce*. Tipe komunikasi kolektif ini, memberikan dua keuntungan yaitu (Hasan, 2011):

1. Operasi komunikasi tersebut dapat digunakan untuk mengekspresikan operasi yang kompleks dengan menggunakan semantik yang sederhana
2. Implementasi dapat melakukan pengoptimasian operasi melalui cara yang tidak disediakan oleh tipe operasi komunikasi *point-to-point*.

BAB III
METODOLOGI PENELITIAN DAN PERANCANGAN DIAGRAM ALIR (*FLOWCHART*)

3.1 Metodologi Penelitian

Pada penelitian ini, kompilasi dan eksekusi kode program paralel diuji pada jaringan fisik. Tahapan pertama yang perlu dilakukan adalah membangun jaringan fisik dengan sistem operasi *Linux Pelicanhpc-v2.2* berbasis *Debian.* Dalam penelitian ini dipasang satu *head node* dan dua *compute node.*

Head node merupakan komputer yang digunakan sebagai tempat bagi pengguna *cluster* untuk mengakses *cluster* sehingga pengguna dapat memberikan tugas-tugas komputasi ke dalam *cluster*. *Head node* harus dapat diakses melalui jaringan, karena *head node* adalah tempat mengakses dan memberikan tugas ke *compute node.*

Compute node adalah komputer pekerja yang menerima tugas dari *head node* dan memproses tugas tersebut. Komputer yang difungsikan sebagai *compute node* hanya dapat diakses oleh *head node* untuk melakukan proses komputasi.

Adapun spesifikasi *head node* dan *compute node* seperti terlihat pada Tabel 3.1.

Tabel 3.1 Spesifikasi *head node* dan *compute node*

Perangkat	***Head Node***	***Compute Node 1***	***Compute Node 2***
Hard disk	320 gigabyte	320 gigabyte	320 gigabyte
RAM	1 gigabyte	2 gigabyte	2 gigabyte
Prosesor	*Intel(R) Core(TM) i3 CPU*	*Intel(R) Core(TM) i3-2328M CPU*	*Intel(R) Atom(TM) CPU*
Sistem Operasi	*Windows* 7 *Ultimate* (32 bit)	*Windows* 7 *Ultimate* (32 bit)	*Windows* 7 *Ultimate* (32 bit)

Kemudian setelah *compute node* dan *head node* dipasang pada masing-masing komputer yaitu menghubungkan *compute node 1*, *compute node 2* dan *head node* ke *switch* menggunakan kabel *straight-through* untuk pembuktian jaringan fisik telah berjalan dengan baik.

Dalam penelitian ini kompilasi dan eksekusi kode program paralel dilakukan pada sistem operasi *Linux PelicanHPC* berbasis *Debian.* Perintah kompilasi dan eksekusi pada kode paralel C dan C++ seperti terlihat pada Tabel 3.2.

Tabel 3.2 Perintah kompilasi dan eksekusi kode paralel C dan C++

	Kode paralel C	**Kode paralel C++**
Kompilasi	*mpicc –o contoh contoh.c*	*mpiCC –o ccontoh2 ccontoh.cc*
Eksekusi	*mpirun –np 2 contoh*	*mpirun –np 2 ccontoh*

Dalam penelitian ini, ada delapan kode paralel C dan kode paralel C++ yang perlu dilakukan perbandingan waktu eksekusi tersebut. Analisis waktu yang dilakukan berdasarkan waktu rata-rata dari 10 percobaan.

3.2 Perancangan Diagram Alir *(Flowchart)*

Diagram alir *(flowchart)* merupakan gambar atau bagan yang memperlihatkan urutan dan hubungan antar proses beserta instruksinya. Gambaran ini dinyatakan dengan simbol. Dengan demikian setiap simbol menggambarkan proses tertentu, sedangkan antara proses digambarkan dengan garis penghubung. Pada subbab ini dibuat delapan *flowchart* pemrograman paralel pada rutin komunikasi kolektif yang terdiri dari:

1. *Flowchart* program sinkronisasi *barrier*
2. *Flowchart* program *broadcast*
3. *Flowchart* program *gather* dan *scatter*
4. *Flowchart* program *gather-to-all*
5. *Flowchart* program *all-to-all scatter* dan *gather*
6. *Flowchart* program *reduce*
7. *Flowchart* program *reduce-scatter*
8. *Flowchart* program *scan*

3.2.1 *Flowchart* Program Sinkronisasi *Barrier*

Gambar 3.1 adalah *flowchart* dari program sinkronisasi *Barrier* yang digunakan untuk melakukan komputasi sederhana, yaitu proses perkalian berdasarkan nilai *rank* yang diperoleh oleh proses.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat proses *rank* kemudian ditampilkan ke layar. Berikutnya dilakukan komputasi sederhana yaitu proses perkalian berdasarkan nilai *rank* yang diperoleh oleh proses. Selanjutnya memanggil *MPI_Barrier* untuk menghalangi semua proses. Setelah itu ditampilkan total data pada *rank* ke layar.

Langkah terakhir pada *flowchart* program sinkronisasi *Barrier* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.3 Baris program Sinkronisasi *Barrier* bahasa C dan C++

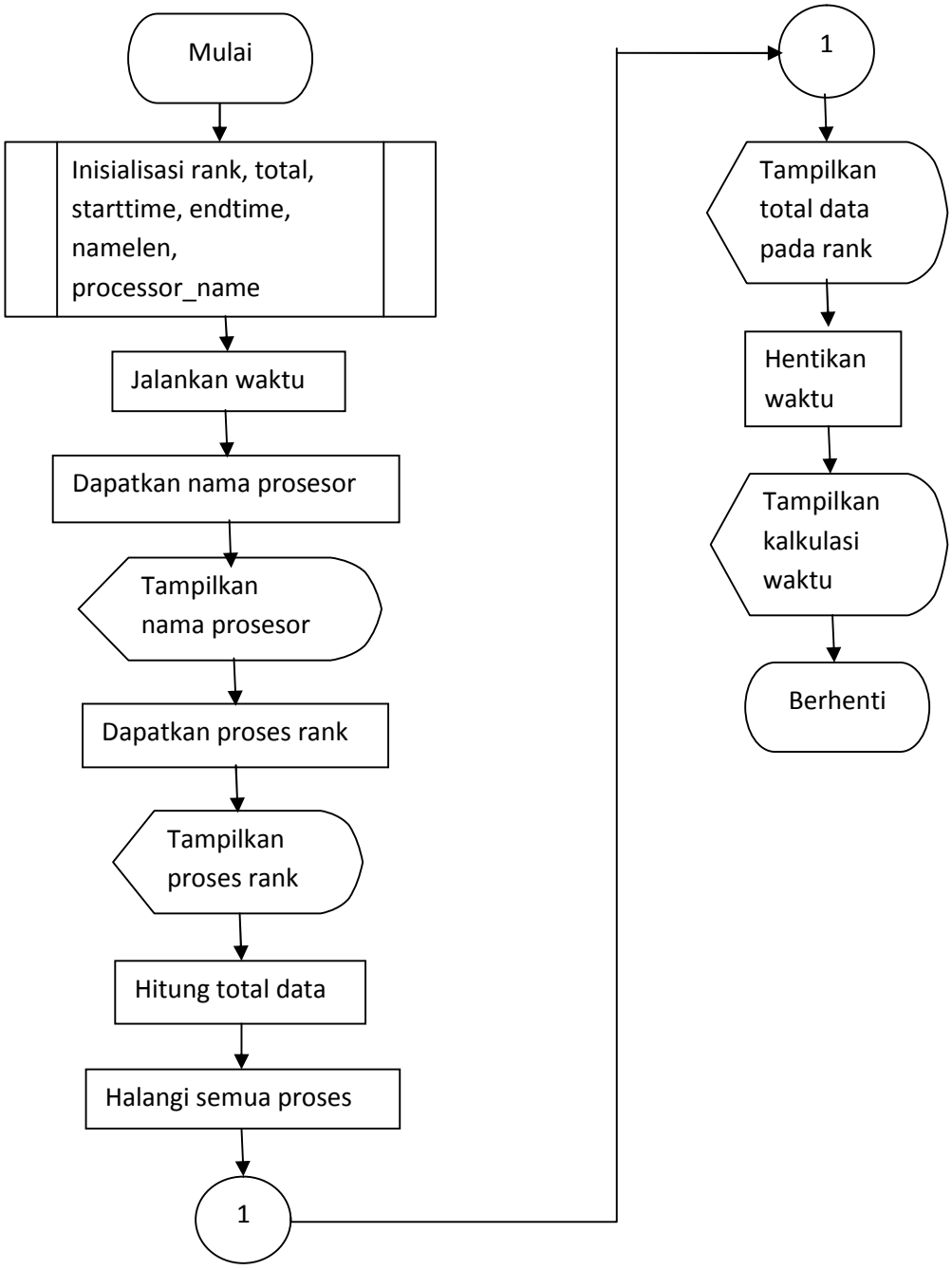
Bahasa C	**Bahasa C++**
total = (rank+1)*25; MPI_Barrier(MPI_COMM_WORLD); printf("Total data pada rank %d: %d\r\n",rank,total);	total = (rank+1)*25; MPI::COMM_WORLD.Barrier(); printf("Total data pada rank %d: %d\r\n",rank,total);

Tabel 3.3 merupakan potongan baris program Sinkronisasi *Barrier* yang bertujuan menahan semua proses yang terjadi sampai seluruh proses telah mencapai rutinnya. Proses komputasi yang terjadi yaitu:

```
total = (rank+1)*25;
```

Selanjutnya dipanggil *MPI_Barrier* untuk *blocking* semua proses, kemudian semua hasil ditampilkan. Proses yang terjadi yaitu:

```
MPI_Barrier(MPI_COMM_WORLD);
printf("Total data pada rank %d:     %d\r\n",rank,total);
```



Gambar 3.1 *Flowchart* program sinkronisasi *Barrier*

3.2.2 *Flowchart* Program *Broadcast*

Gambar 3.2 adalah *flowchart* dari program *Broadcast* untuk menyiarkan data, jumlah data, tipe data, dan *rank* dari suatu proses dalam suatu spesifik *group* ke seluruh proses pada spesifik *group*.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat proses *rank* kemudian ditampilkan ke layar. Berikutnya jika kondisi *rank 0* maka inisialisasi *val=100,* kemudian disiarkan data, jumlah data, tipe data dan *rank*. Setelah itu ditampilkan *rank* dan *val* ke layar. Jika tidak maka langsung ditampilkan *rank* dan *val* saja.

Langkah terakhir pada *flowchart* program *Broadcast* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.4 Baris program *Broadcast* bahasa C dan C++

Bahasa C	**Bahasa C++**

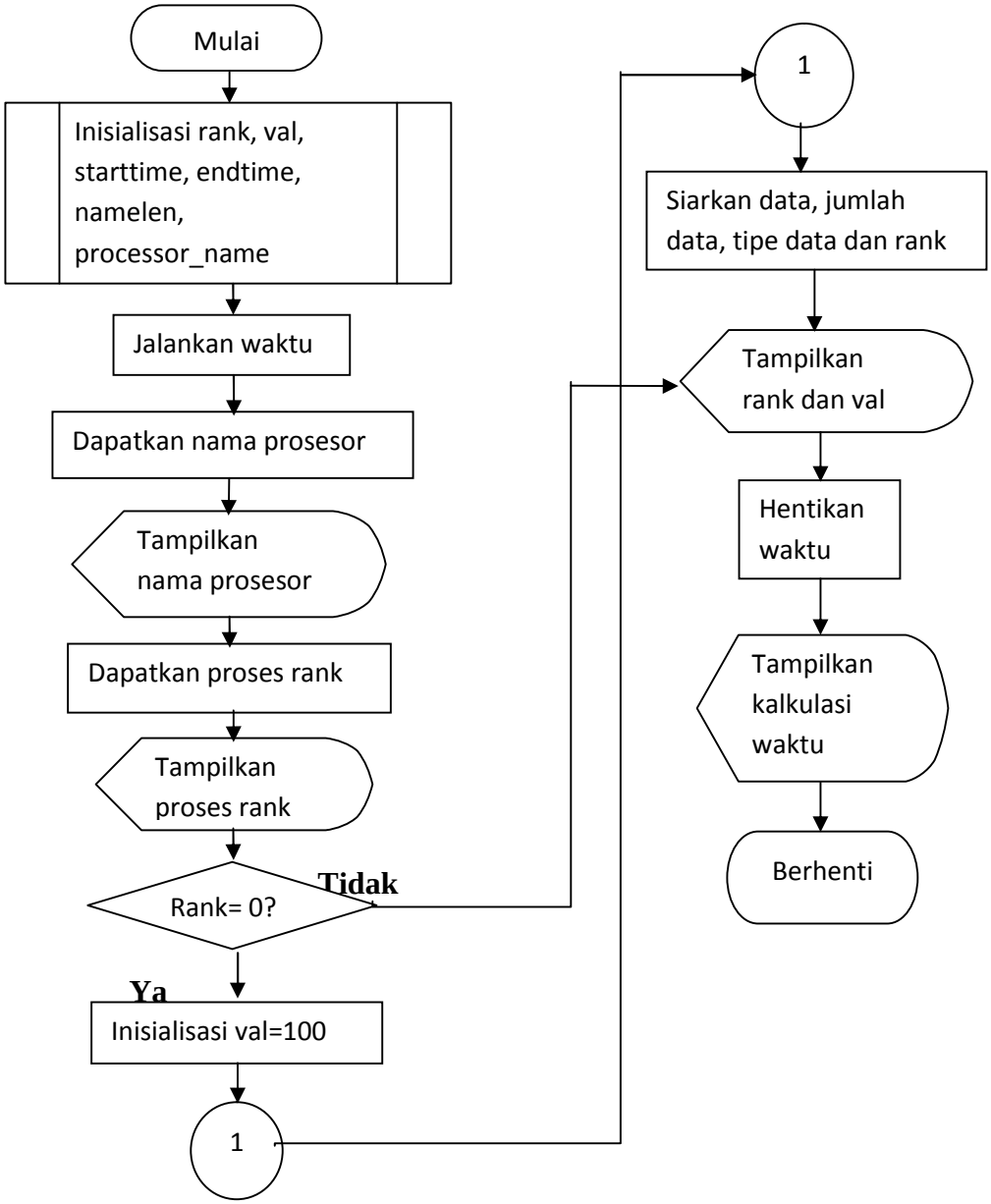
if(rank==0)
val = 100;
MPI_Bcast(&val, 1, MPI_INT, 0,
MPI_COMM_WORLD);
printf("Rank %d, Total val =
%d\r\n",rank,val);
	if(rank==0)
	val = 100;
	MPI::COMM_WORLD.Bcast(&va
	l, 1, MPI::INT, 0);
	printf("Rank %d, Total val =
	%d\r\n",rank,val);

Tabel 3.4 merupakan potongan baris program *Broadcast* yang bertujuan untuk menyiarkan pesan dari proses dengan *rank* ke seluruh proses lain pada spesifik *group.* Pada *rank* 0, dilakukan inisialisasi nilai *val:*

```
if(rank==0)
        val = 100;
```

Selanjutnya pada *rank* 0 dilakukan *Broadcast* pada *communicator MPI_COMM_WORLD.* Untuk *rank* bukan 0 ditampilkan hasil data yang dikirim oleh *rank* 0. Baris program yang digunakan adalah:

```
MPI_Bcast( &val, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf("Rank %d, Total val = %d\r\n",rank,val);
```



Gambar 3.2 *Flowchart* program *Broadcast*

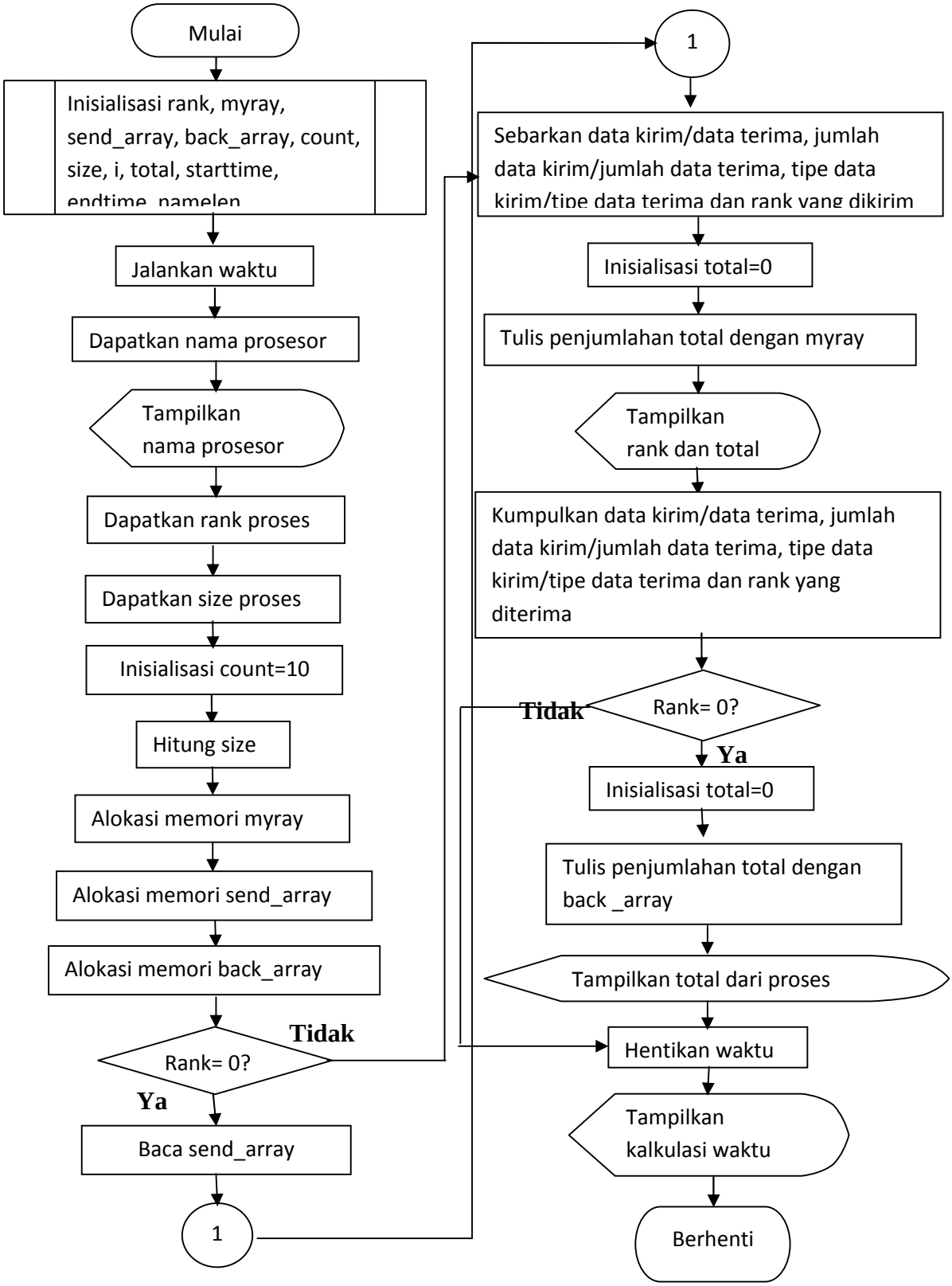
3.2.3 *Flowchart* Program *Gather* dan *Scatter*

Gambar 3.3 adalah *flowchart* dari program *Gather* dan *Scatter* untuk menyebarkan dan mengumpulkan data kirim atau data terima, jumlah data kirim atau jumlah data terima, tipe data kirim atau tipe data terima dan *rank* yang dikirim atau *rank* yang telah diterima.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat *rank* proses dan *size* proses, kemudian inisialisasi *count=10* dan dihitung *size.* Selanjutnya dialokasikan memori *myray, send_array, back_array.*

Pada *flowchart* program *gather* dan *scatter* terdapat dua kondisi yaitu *rank 0* sebagai pengirim data dan *rank 0* sebagai penerima data. Jika *rank 0* sebagai pengirim data maka dibaca *send_array,* kemudian disebarkan data kirim atau data terima, jumlah data kirim atau jumlah data terima, tipe data kirim atau tipe data terima dan *rank* yang dikirim. Setelah itu inisialisasi total=0, kemudian ditulis penjumlahan total dengan *myray.* Kemudian ditampilkan *rank* dan total, selanjutnya dikumpulkan data kirim atau data terima, jumlah data kirim atau jumlah data terima, tipe data kirim atau tipe data terima dan *rank* yang telah diterima. Jika *rank 0* sebagai penerima data maka inisialisasi total=0, kemudian ditulis penjumlahan total dengan *back_array*. Selanjutnya ditampilkan total dari proses.

Langkah terakhir pada *flowchart* program *Gather* dan *Scatter* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.



Gambar 3.3 *Flowchart* program *Gather* dan *Scatter*

Tabel 3.5 Baris program *Gather* dan *Scatter* bahasa C dan C++

Bahasa C	**Bahasa C++**

count=10;
size=count*numnodes;
myray=(int*)malloc(count*sizeof(int));
send_array=(int*)malloc(size*sizeof(int));
back_array=(int*)malloc(numnodes*sizeof(int));
if(rank==0)
{ for(i=0;i<size;i++)
send_array[i]=i;
}
MPI_Scatter(send_array,
count,MPI_INT,myray,
count,MPI_INT, 0,
MPI_COMM_WORLD);
total=0;
for(i=0;i<count;i++)
total=total+myray[i];
printf("Rank= %d Total= %d
\r\n",rank,total);
MPI_Gather(&total, 1, MPI_INT,
back_array, 1, MPI_INT, 0,
MPI_COMM_WORLD);
if(rank == 0)
{ total=0;
for(i=0;i<numnodes;i++)
total = total + back_array[i];
printf("Total dari proses= %d \r\n
",total); }
	count=10;
	size=count*numnodes;
	myray=(int*)malloc(count*sizeof(int));
	send_array=(int*)malloc(size*sizeof(int));
	back_array=(int*)malloc(numnodes*sizeof(int));
	if(rank==0)
	{ for(i=0;i<size;i++)
	send_array[i]=i;
	}
	MPI::COMM_WORLD.Scatter(send_array, count,MPI::INT,myray,
	count,MPI::INT,0);
	total=0;
	for(i=0;i<count;i++)
	total=total+myray[i];
	printf("Rank= %d Total= %d
	\r\n",rank,total);
	MPI::COMM_WORLD.Gather(&total,
	1, MPI::INT, back_array, 1, MPI::INT,
	0);
	if(rank == 0)
	{ total=0;
	for(i=0;i<numnodes;i++)
	total = total + back_array[i];
	printf("Total dari proses= %d \r\n
	",total);
	}

Tabel 3.5 merupakan potongan baris program *Gather* dan *Scatter* yang bertujuan mengumpulkan nilai bersama dari grup proses dan mengirim data dari satu task ke seluruh task lain dalam grup. Disiapkan data untuk proses *Scatter* di mana jumlah data adalah 10 kali jumlah proses.

```
count=10;
size=count*numnodes;
myray=(int*)malloc(count*sizeof(int));
send_array=(int*)malloc(size*sizeof(int));
back_array=(int*)malloc(numnodes*sizeof(int));
if(rank==0)
   {
        for(i=0;i<size;i++)
        send_array[i]=i;
    }
```

Selanjutnya dilakukan proses *MPI_Scatter* pada *rank* 0.

```
MPI_Scatter(send_array, count,MPI_INT,myray, count,MPI_INT,  0,
MPI_COMM_WORLD);
```

Data yang diterima selanjutnya dijumlahkan dan ditampilkan ke layar.

```
total=0;
for(i=0;i<count;i++)
total=total+myray[i];
printf("Rank= %d Total= %d \r\n",rank,total);
```

Selanjutnya nilai total menjadi input data pada proses *MPI_Gather.*

```
MPI_Gather(&total, 1, MPI_INT, back_array, 1,  MPI_INT, 0,
MPI_COMM_WORLD);
```

Di sini penerima data ada pada *rank* 0 sehingga dilakukan proses penjumlahan semua data yang diterima pada *rank* 0.

```
if(rank == 0)
{       total=0;
        for(i=0;i<numnodes;i++)
        total = total + back_array[i];
printf("Total dari proses= %d \r\n ",total); }
```

3.2.4 *Flowchart* Program *Gather-to-All*

Gambar 3.4 adalah *flowchart* dari program *Gather-to-All*, program *Gather-to-All* sama seperti *MPI_Gather*. Pada proses *MPI_Gather* hanya pada *rank* tertentu yang menjadi penerima data, tetapi pada program *Gather-to-All* semua proses akan menerima data.

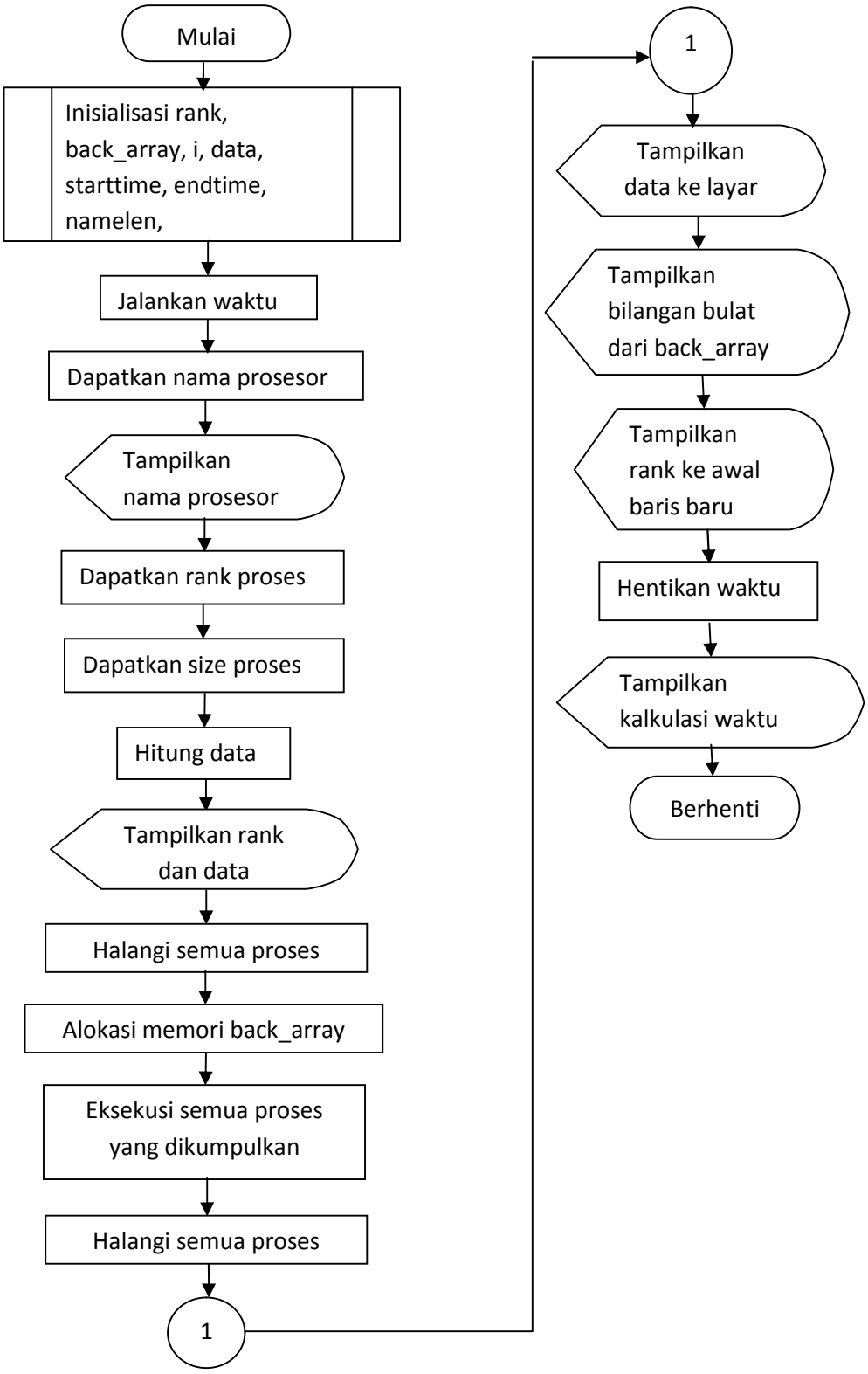
Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat *rank* proses dan *size* proses, kemudian dihitung data. Selanjutnya ditampilkan *rank* dan data kemudian halangi semua proses. Berikutnya dialokasikan memori *back_array* dan dihalang semua proses, kemudian ditampilkan data ke layar. Selanjutnya ditampilkan bilangan bulat dari *back_array*, kemudian ditampilkan *rank* ke awal baris baru.

Langkah terakhir pada *flowchart* program *Gather-to-All* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.6 Baris program *Gather-to-All* bahasa C dan C++

Bahasa C	**Bahasa C++**

data = rank + 1;
printf("Rank= %d, Data= %d \r\n",rank,data);
MPI_Barrier(MPI_COMM_WORLD);
back_array=(int*)malloc(numnodes *sizeof(int));
MPI_Allgather(&data, 1, MPI_INT,
back_array, 1, MPI_INT,
MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
printf("Gather>> ");
for(i=0;i<numnodes;i++)
printf("%d ",back_array[i]);
printf("\r\n");
	data = rank + 1;
	printf("Rank= %d, Data= %d \r\n",rank,data);
	MPI::COMM_WORLD.Barrier();
	back_array = (int*)malloc(numnodes*sizeof(int));
	MPI::COMM_WORLD.Allgather(&data, 1, MPI::INT, back_array, 1, MPI::INT);
	MPI::COMM_WORLD.Barrier();
	printf("Gather>> ");
	for(i=0;i<numnodes;i++)
	printf("%d ",back_array[i]);
	printf("\r\n");



Gambar 3.4 *Flowchart* program *Gather-to-All*

Tabel 3.6 merupakan potongan baris program *Gather-to-All* yang bertujuan untuk mengumpulkan data dari seluruh task dan mendistribusikannya untuk semua. Disiapkan data untuk proses *MPI Gather-to-All.* Di sini ada *MPI_Barrier* supaya memastikan persiapan data mencapai posisi yang sama sebelum eksekusi *MPI Gather-to-All.*

```
data = rank + 1;
printf("Rank= %d, Data= %d \r\n",rank,data);
MPI_Barrier(MPI_COMM_WORLD);
back_array=(int*)malloc(numnodes*sizeof(int));
```

Selanjutnya dieksekusi *MPI Gather-to-All* dan memanggil *MPI_Barrier* untuk memastikan proses sudah menerima data. Selanjutnya ditampilkan data ke layar.

```
MPI_Allgather(&data, 1, MPI_INT,
back_array, 1,  MPI_INT,
MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
printf("Gather>> ");
for(i=0;i<numnodes;i++)
printf("%d ",back_array[i]);
 printf("\r\n");
```

3.2.5 *Flowchart* Program *All-to-All Scatter dan Gather*

Gambar 3.5 adalah *flowchart* dari program *All-to-All Scatter* dan *Gather* merupakan pengembangan proses *MPI Gather-to-all*, tetapi pengguna dapat memilih data yang dikirim.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat *rank* proses dan *size* proses, kemudian dialokasikan memori *data_send* dan *data_recv.* Selanjutnya diacak *rank* secara statis, kemudian dibaca *num* dan ditulis *data_send.* Setelah itu

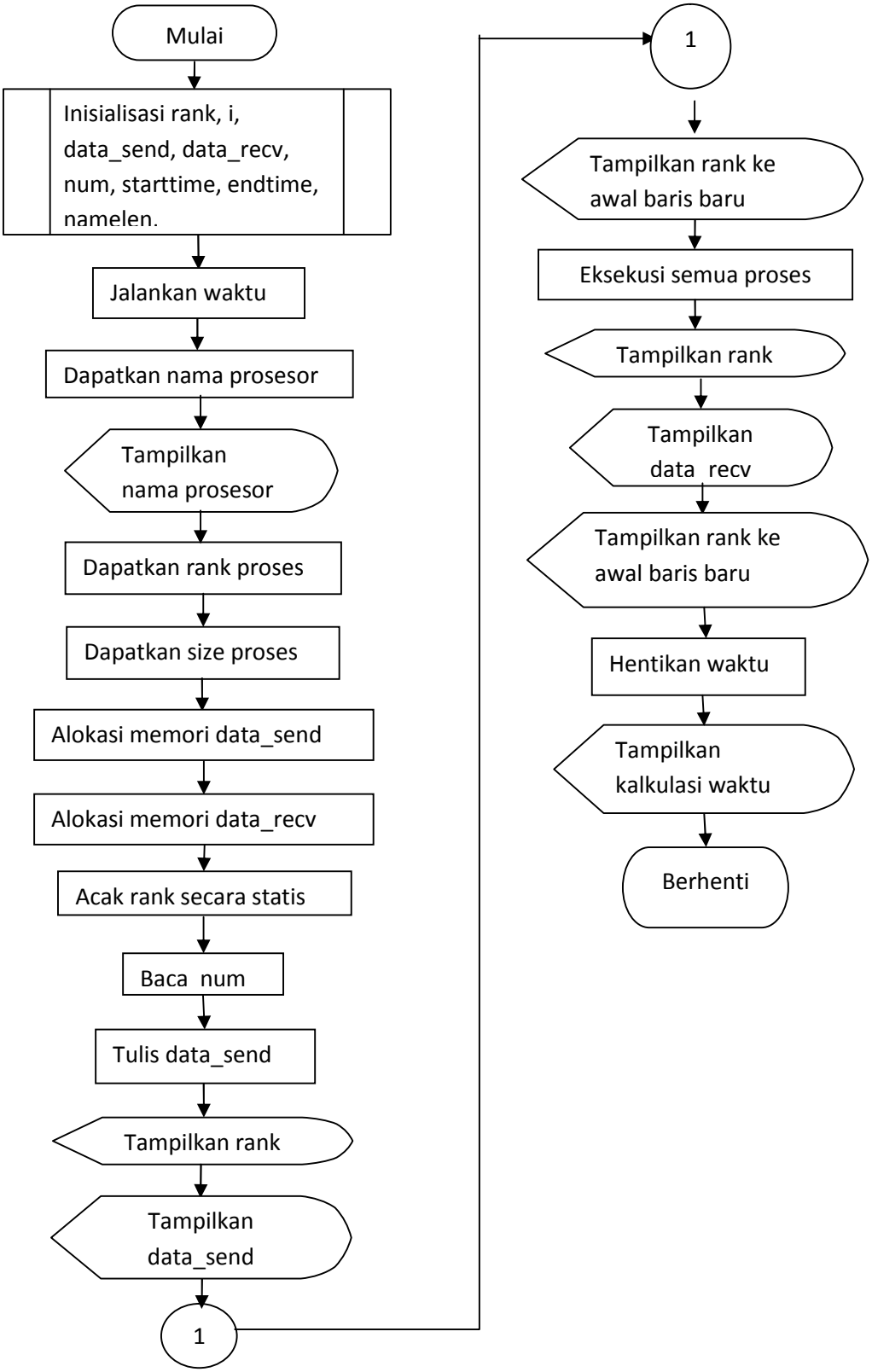
ditampilkan *rank*, ditampilkan *data_send* dan ditampilkan *rank* ke awal baris baru. Selanjutya dieksekusi semua proses, kemudian ditampilkan *rank* ditampilkan *data_recv* dan ditampilkan *rank* ke awal baris baru.

Langkah terakhir pada *flowchart* program *All-to-All Scatter* dan *Gather* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.7 Baris program *All-to-All Scatter* dan *Gather* bahasa C dan C++

Bahasa C	**Bahasa C++**

data_send=(int*)malloc(sizeof(int)*numnodes);
data_recv=(int*)malloc(sizeof(int)*numnodes);
srand(rank);
for(i=0;i<numnodes;i++)
{
num = (float)rand()/RAND_MAX;
data_send[i]=(int)(10.0*num)+1;
}
printf("Rank= %d Data dikirim =
",rank);
for(i=0;i<numnodes;i++)
printf("%d ",data_send[i]);
printf("\r\n");
 	
MPI_Alltoall(data_send,1,MPI_INT,
data_recv,1,MPI_INT,MPI_COMM_
WORLD);
printf("Rank= %d Data diterima =
",rank);
for(i=0;i<numnodes;i++)
printf("%d ",data_recv[i]);
printf("\r\n");
	data_send=(int*)malloc(sizeof(int)*numnodes);
	data_recv=(int*)malloc(sizeof(int)*numnodes);
	srand(rank);
	for(i=0;i<numnodes;i++)
	{
	num = (float)rand()/RAND_MAX;
	data_send[i]=(int)(10.0*num)+1;
	}
	
	printf("Rank= %d Data dikirim =
	",rank);
	for(i=0;i<numnodes;i++)
	printf("%d ",data_send[i]);
	printf("\r\n");
	MPI::COMM_WORLD.Alltoall(data_send,1,MPI::INT,
	data_recv,1,MPI::INT);
	printf("Rank= %d Data diterima =
	",rank);
	for(i=0;i<numnodes;i++)
	printf("%d ",data_recv[i]);
	printf("\r\n");



Gambar 3.5 *Flowchart* program *All-to-All Scatter* dan *Gather*

Tabel 3.7 merupakan potongan baris program *All-to-All Scatter* dan *Gather* yang bertujuan mengirim data dari seluruh proses ke seluruh proses. Disiapkan data untuk proses *MPI All-to-All* yang dilakukan secara acak. Selanjutnya data yang dihasilkan ditampilkan ke layar.

```
data_send=(int*)malloc(sizeof(int)*numnodes);
data_recv=(int*)malloc(sizeof(int)*numnodes);
srand(rank);
for(i=0;i<numnodes;i++)
{
  num = (float)rand()/RAND_MAX;
  data_send[i]=(int)(10.0*num)+1;
 }
printf("Rank= %d Data dikirim  = ",rank);
for(i=0;i<numnodes;i++)
printf("%d ",data_send[i]);
printf("\r\n");
```

Selanjutnya dieksekusi *MPI All-to-All* dan ditampilkan hasilnya ke layar.

```
MPI_Alltoall(data_send,1,MPI_INT,
data_recv,1,MPI_INT,MPI_COMM_WORLD);
printf("Rank= %d Data diterima = ",rank);
for(i=0;i<numnodes;i++)
printf("%d ",data_recv[i]);
printf("\r\n");
```

3.2.6 *Flowchart* Program *Reduce*

Gambar 3.6 adalah *flowchart* dari program *Reduce* untuk melakukan proses pengambilan data seluruh proses dan melakukan operasi matematika seperti penjumlahan, perkalian dan sebagainya.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat *rank* proses dan *size* proses, kemudian disiapkan data penyebaran proses dengan jumlah data 4. Dialokasikan memori myray, kemudian dihitung *size.* Dialokasikan memori *send_data* dan *back_data*.

Pada *flowchart* program *Reduce* terdapat dua kondisi yaitu *rank 0* sebagai pengirim data dan *rank 0* sebagai penerima data. Jika *rank 0* sebagai pengirim data maka dibaca *send_data*, kemudian dieksekusi semua proses yang disebarkan. Inisialisasi penjumlahan total=0, kemudian ditulis penjumlahan total dengan *myray.* Tampilkan *rank* dan total, kemudian lakukan operasi reduksi. Jika *rank 0* sebagai penerima data maka ditampilkan *sum_total.*

Langkah terakhir pada *flowchart* program *Reduce* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.8 Baris program *Reduce* bahasa C dan C++

Bahasa C	**Bahasa C++**

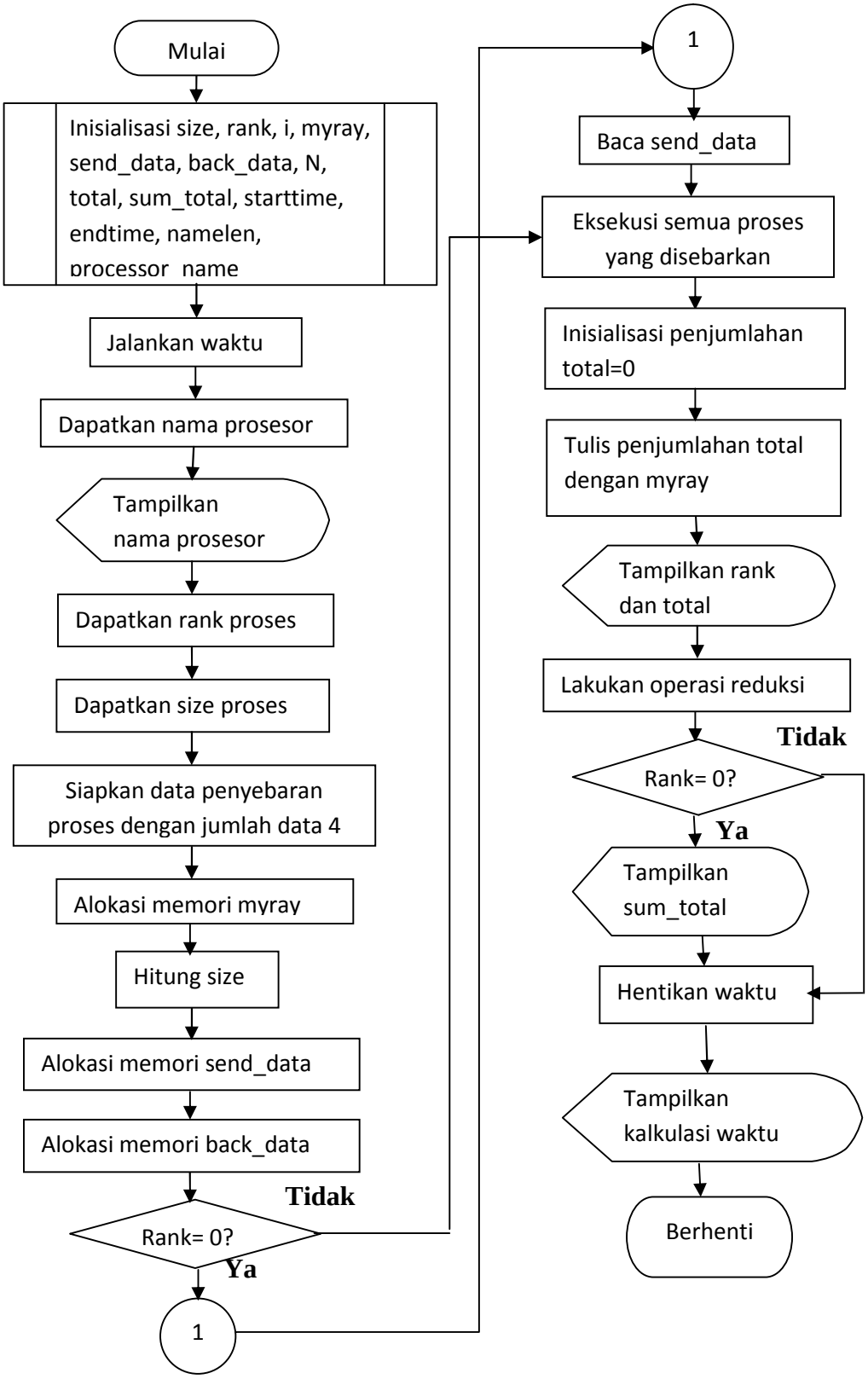
N=4;
myray=(int*)malloc(N*sizeof(int));
size=N*numnodes;
send_data=(int*)malloc(size*sizeof(int));
back_data=(int*)malloc(numnodes*sizeof(int));
if(rank == 0)
{ for(i=0;i<size;i++)
send_data[i]=i;
}
MPI_Scatter(send_data,N,MPI_INT, myray,N,MPI_INT, 0, MPI_COMM_WORLD);
total=0;
for(i=0;i<N;i++)
total=total+myray[i];
printf("rank= %d data= %d\r\n",rank,total);
MPI_Reduce(&total, &sum_total, 1,MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if(rank == 0){
printf("Jumlah total data = %d \r\n",sum_total);
}
	N=4;
	myray=(int*)malloc(N*sizeof(int));
	size=N*numnodes;
	send_data=(int*)malloc(size*sizeof(int));
	back_data=(int*)malloc(numnodes*sizeof(int));
	if(rank == 0)
	{ for(i=0;i<size;i++)
	send_data[i]=i;
	}
	MPI::COMM_WORLD.Scatter(send_data,N,MPI::INT,myray,N,MPI::INT, 0);
	total=0;
	for(i=0;i<N;i++)
	total=total+myray[i];
	printf("rank = %d data = %d\r\n",rank,total);
	MPI::COMM_WORLD.Reduce(&total, &sum_total, 1,MPI::INT, MPI::SUM, 0);
	if(rank == 0)
	{
	printf("Jumlah total data = %d \r\n",sum_total);
	}

Tabel 3.8 merupakan potongan baris program *Reduce* yang bertujuan untuk mengurangi nilai seluruh proses ke nilai tunggal. Disiapkan data untuk proses *MPI Scatter* dengan jumlah data sebanyak 4. Selanjutnya data dieksekusi dengan operasi *MPI Scatter*.

```
N=4;
myray=(int*)malloc(N*sizeof(int));
size=N*numnodes;
send_data=(int*)malloc(size*sizeof(int));
back_data=(int*)malloc(numnodes*sizeof(int));
if(rank == 0)
   {
        for(i=0;i<size;i++)
        send_data[i]=i;
    }
MPI_Scatter(send_data,N,MPI_INT, myray,N,MPI_INT, 0,
MPI_COMM_WORLD);
```

Hasil operasi *MPI Scatter* dilakukan penjumlahan. Data ini menjadi input data operasi *MPI Reduce* dengan operasi penjumlahan. Selain itu, data yang digunakan ditampilkan ke layar.

```
total=0;
for(i=0;i<N;i++)
   total=total+myray[i];
printf("rank= %d data= %d\r\n",rank,total);
MPI_Reduce(&total, &sum_total, 1,MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
if(rank == 0)
{
printf("Jumlah total data = %d \r\n",sum_total);
}
```



Gambar 3.6 *Flowchart* program *Reduce*

3.2.7 *Flowchart* Program *Reduce-Scatter*

Gambar 3.7 adalah *flowchart* dari program *Reduce-Scatter* untuk melakukan penyebaran hasil operasi reduksi ke semua proses yang ada.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat *size* proses dan *rank* proses, kemudian dialokasikan memori *send_data* dan *recvcount.* Ditampilkan *rank*, kemudian diacak *rank* secara statis. Selanjutnya dibaca *num* dan ditulis *send_data*, kemudian dibaca *recvcount.* Ditampilkan *send_data*, dan ditampilkan *rank* ke awal baris baru. Selanjutnya eksekusi semua proses reduksi yang disebarkan dengan operasi penjumlahan, kemudian ditampilkan *rank* dan *recvbuf.*

Langkah terakhir pada *flowchart* program *Reduce-Scatter* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.9 merupakan potongan baris program *Reduce-Scatter* yang bertujuan untuk menggabungkan nilai dan menyebarkan hasilnya. Disiapkan data untuk proses *MPI Reduce-Scatter* dan ditampilkan ke layar.

```
send_data = (int *) malloc(size* sizeof(int));
recvcounts = (int *)malloc(size* sizeof(int));
printf("Rank %d send_data= ",rank);
srand(rank); for (i=0; i<size; i++)
{
num = (float)rand()/RAND_MAX;        send_data[i] = (int)(10.0*num)+1;
recvcounts[i] = 1;
printf("%d ",send_data[i]);
}
printf("\r\n");
```

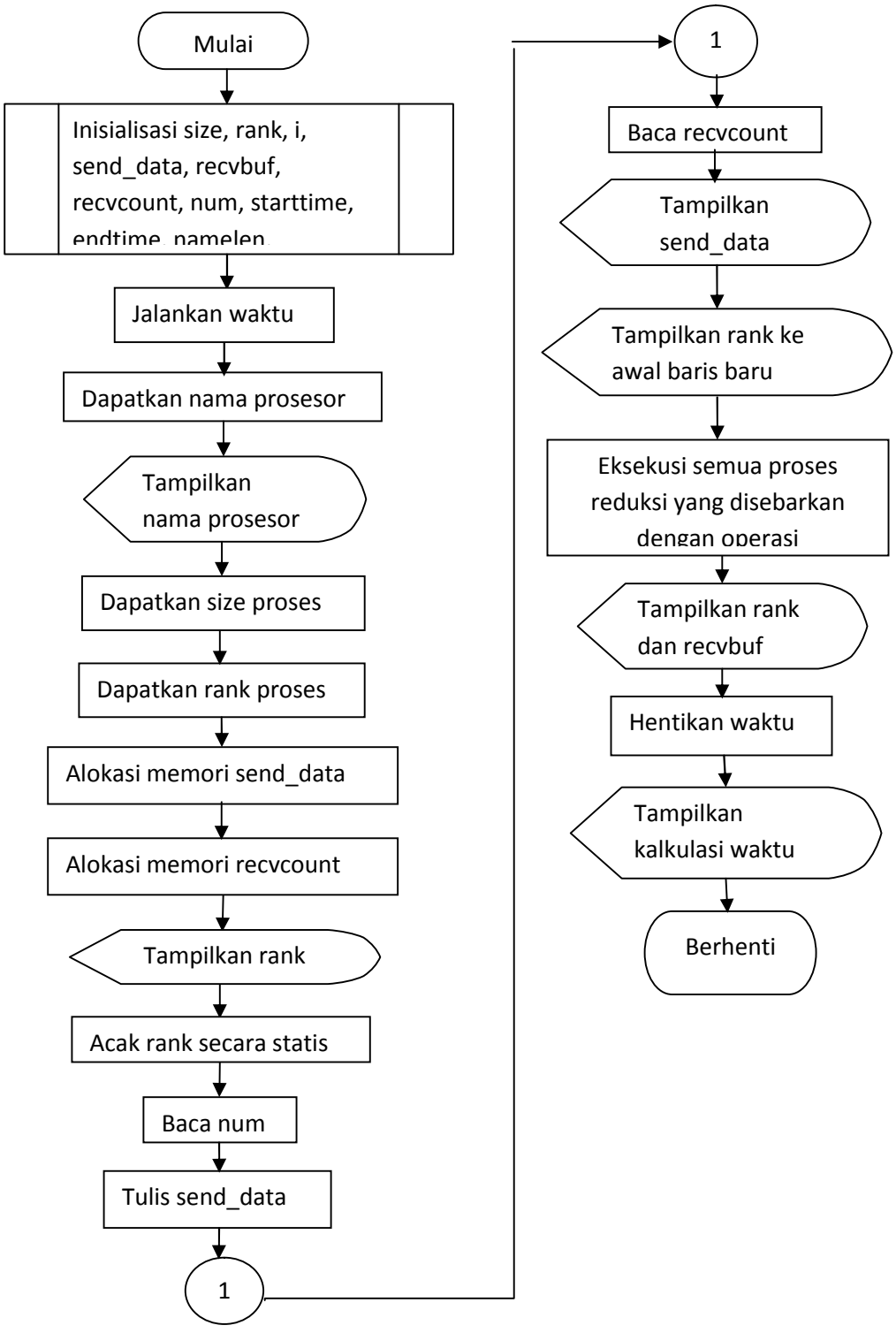
Selanjutnya dieksekusi *MPI Reduce-Scatter* dengan operasi penjumlahan. Hasil eksekusi ditampilkan ke layar.

```
MPI_Reduce_scatter(send_data, &recvbuf, recvcounts, MPI_INT, MPI_SUM,
MPI_COMM_WORLD);
printf("Rank %d recvbuf= %d\r\n",rank,recvbuf);
```

Tabel 3.9 Baris program *Reduce-Scatter* bahasa C dan C++

Bahasa C	**Bahasa C++**

send_data = (int *) malloc(size* sizeof(int));
recvcounts = (int *)malloc(size* sizeof(int));
printf("Rank %d send_data= ",rank);
srand(rank); for (i=0; i<size; i++)
{
num = (float)rand()/RAND_MAX;
send_data[i] = (int)(10.0*num)+1;
recvcounts[i] = 1;
printf("%d ",send_data[i]);
}
printf("\r\n");
MPI_Reduce_scatter(send_data, &recvbuf, recvcounts, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
printf("Rank %d recvbuf= %d\r\n",rank,recvbuf);
	send_data = (int *) malloc(size* sizeof(int));
	recvcounts = (int *)malloc(size* sizeof(int));
	printf("Rank %d send_data= ",rank);
	srand(rank); for (i=0; i<size; i++)
	{
	num = (float)rand()/RAND_MAX;
	send_data[i] = (int)(10.0*num)+1;
	recvcounts[i] = 1;
	printf("%d ",send_data[i]);
	}
	printf("\r\n");
	MPI::COMM_WORLD.Reduce_scatter (send_data, &recvbuf, recvcounts, MPI::INT, MPI::SUM);
	printf("Rank %d recvbuf = %d\r\n",rank,recvbuf);



Gambar 3.7 *Flowchart* program *Reduce-Scatter*

3.2.8 *Flowchart* Program *Scan*

Gambar 3.8 adalah *flowchart* dari program *Scan* untuk memindai semua proses yang terdapat di lingkungan bukan intracommunicator.

Langkah pertama yaitu program dimulai, kemudian dimasukkan inisialisasi yang dibuat. Selanjutnya dijalankan waktu eksekusi program, kemudian didapat nama prosesor dan ditampilkan ke layar. Selanjutnya didapat *size* proses dan *rank* proses, kemudian dihitung data. Selanjutnya tampilkan *rank* dan data, kemudian eksekusi semua proses yang dipindai. Setelah itu ditampilkan *rank* dan *result*.

Langkah terakhir pada *flowchart* program *Scan* yaitu waktu dihentikan, kemudian ditampilkan kalkulasi waktu dan program diakhiri.

Tabel 3.10 Baris program *Scan* bahasa C dan C++

Bahasa C	**Bahasa C++**

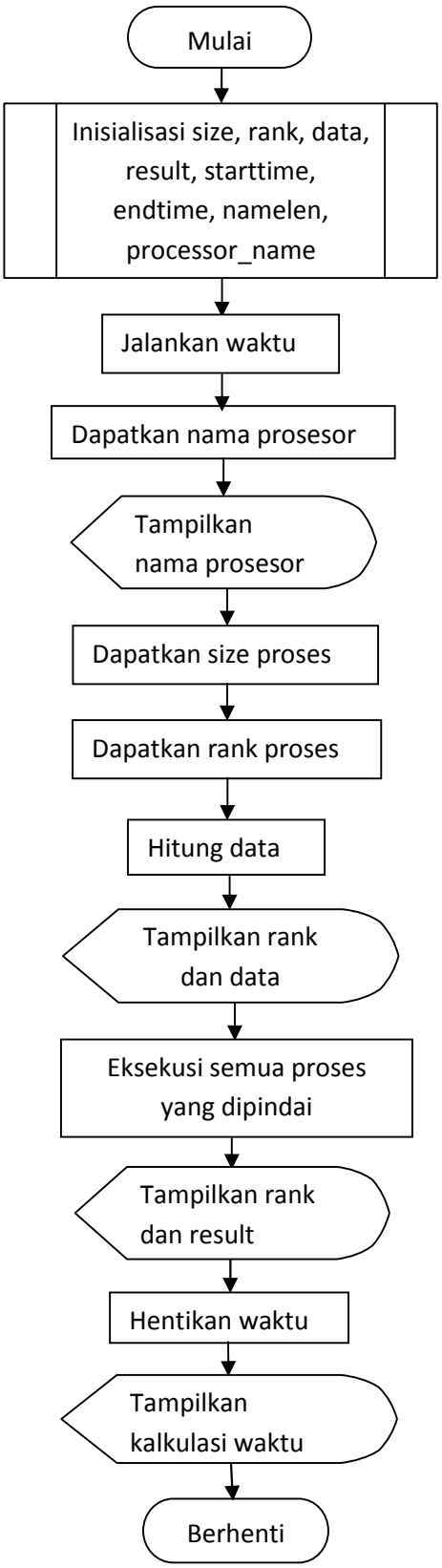
data = rank + 1;
printf("Rank %d data = %d\r\n",rank,data);
MPI_Scan (&data, &result, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
printf("Rank %d result = %d\r\n",rank,result);
	data = rank + 1;
	printf("Rank %d data = %d\r\n",rank,data);
	MPI::COMM_WORLD.Scan(&data, &result, 1, MPI::INT, MPI::SUM);
	printf("Rank %d result = %d\r\n",rank,result);

Tabel 3.10 merupakan potongan baris program *Scan* yang bertujuan untuk menghitung pindaian data koleksi proses. Disiapkan data dan ditampilkan ke layar.

```
data = rank + 1;
printf("Rank %d data = %d\r\n",rank,data);
```

Selanjutnya dieksekusi *MPI Scan* dan hasilnya ditampilkan ke layar.

```
MPI_Scan (&data, &result, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
printf("Rank %d result = %d\r\n",rank,result);
```



Gambar 3.8 *Flowchart* program *Scan*

BAB IV
ANALISIS KODE PROGRAM

Setelah melakukan tahap perancangan jaringan fisik, dilakukan pengujian untuk pembuktian bahwa jaringan fisik telah berjalan dengan baik. Kemudian dilakukan kompilasi dan eksekusi kode program paralel, tahap selanjutnya yang perlu dilakukan adalah membandingkan waktu eksekusi antara pemrograman paralel terstruktur dengan pemrograman paralel berorientasi objek pada rutin komunikasi kolektif. Analisis waktu berdasarkan waktu rata-rata dari 10 percobaan.

4.1 Program Sinkronisasi *Barrier*

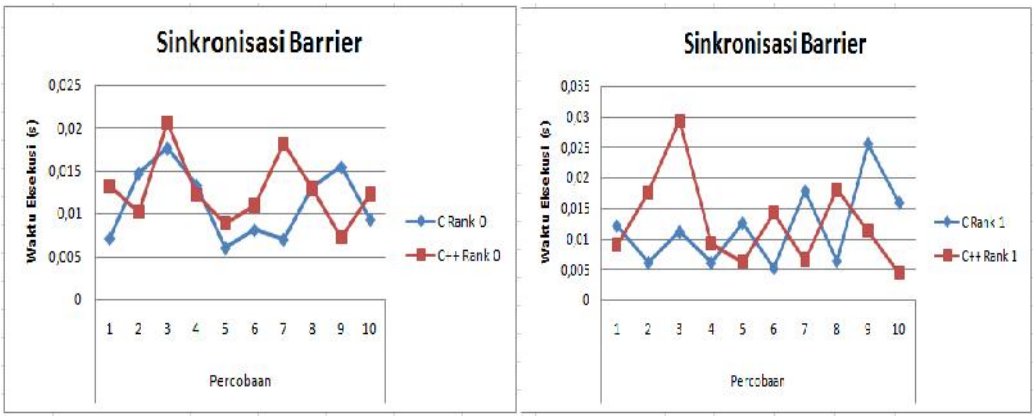
Berdasarkan Tabel 4.1 dan Tabel 4.2 diperoleh rata-rata waktu eksekusi program Sinkronisasi *Barrier* yang telah di-*compile* pada saat *rank* 0 dan *rank* 1 ditahan, waktu eksekusi bahasa C++ lebih cepat daripada bahasa C. Dari Tabel 4.1 dapat dibuat grafik perbandingan waktu eksekusi program Sinkronisasi *Barrier* antara bahasa C dan bahasa C++ pada saat *rank* 0 ditahan seperti terlihat pada Gambar 4.1. (a). Dan dari Tabel 4.2 dapat dibuat grafik perbandingan waktu eksekusi program Sinkronisasi *Barrier* antara bahasa C dan bahasa C++ pada saat *rank* 1 ditahan seperti terlihat pada Gambar 4.1. (b).

Tabel 4.1 Waktu eksekusi program Sinkronisasi *Barrier* pada *rank* 0

Bahasa	Percobaan										Rata-rata
	1	2	3	4	5	6	7	8	9	10	waktu (s)
C	0,007	0,015	0,02	0,013	0,006	0,008	0,007	0,013	0,015	0,009	0,0113
C++	0,013	0,01	0,02	0,012	0,009	0,01	0,018	0,013	0,007	0,012	0,0124
Selisih											-0,0011

Tabel 4.2 Waktu eksekusi program Sinkronisasi *Barrier* pada *rank* 1

Bahasa	Percobaan										Rata-rata
	1	2	3	4	5	6	7	8	9	10	waktu (s)
C	0,012	0,006	0,011	0,006	0,012	0,005	0,02	0,006	0,025	0,016	0,0119
C++	0,009	0,017	0,03	0,009	0,006	0,014	0,006	0,018	0,011	0,004	0,0124
Selisih											-0,0005



(a) (b)

Gambar 4.1 Grafik hasil eksekusi program Sinkronisasi *Barrier* pada (a) *rank* 0, (b) *rank* 1

4.2 Program *Broadcast*

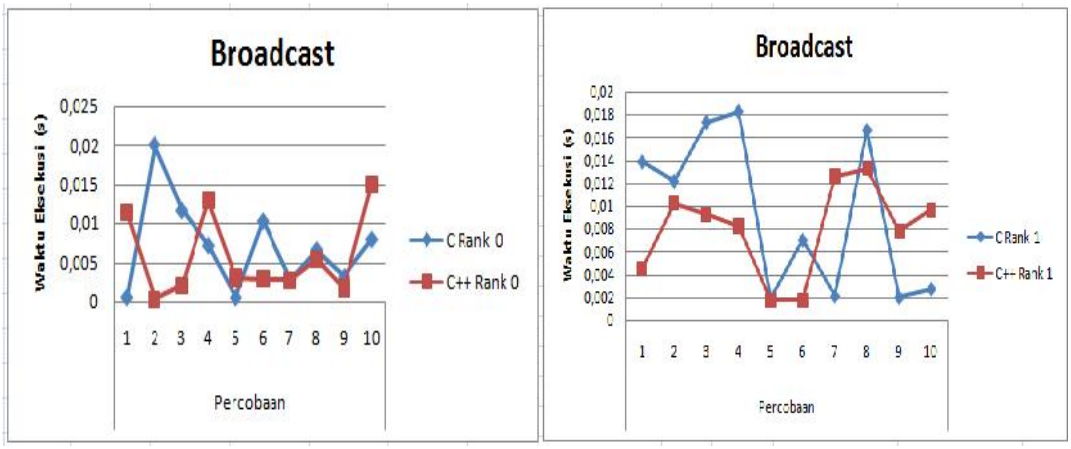
Berdasarkan Tabel 4.3 dan Tabel 4.4 diperoleh rata-rata waktu eksekusi program *Broadcast* yang telah di-*compile* pada saat *rank* 0 dan *rank* 1 disiarkan, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Dari Tabel 4.3 dapat dibuat grafik perbandingan waktu eksekusi program *Broadcast* antara bahasa C dan bahasa C++ pada saat *rank* 0 disiarkan seperti terlihat pada Gambar 4.2. (a). Dan dari Tabel 4.4 dapat dibuat grafik perbandingan waktu eksekusi program *Broadcast* antara bahasa C dan bahasa C++ pada saat *rank* 1 disiarkan seperti terlihat pada Gambar 4.2. (b).

Tabel 4.3 Waktu eksekusi program *Broadcast* pada *rank* 0

Bahasa	Percobaan										Rata-rata
	1	2	3	4	5	6	7	8	9	10	waktu (s)
C	0,001	0,02	0,012	0,007	0,0005	0,01	0,003	0,007	0,003	0,008	0,00715
C++	0,011	0,0003	0,002	0,013	0,003	0,003	0,003	0,005	0,002	0,015	0,00573
Selisih											0,00142

Tabel 4.4 Waktu eksekusi program *Broadcast* pada *rank* 1

Bahasa	Percobaan										Rata-rata
	1	2	3	4	5	6	7	8	9	10	waktu (s)
C	0,014	0,012	0,017	0,018	0,002	0,007	0,002	0,016	0,002	0,003	0,0093
C++	0,004	0,01	0,009	0,008	0,002	0,002	0,012	0,013	0,008	0,009	0,0077
Selisih											0,0016



(a) (b)

Gambar 4.2 Grafik hasil eksekusi program *Broadcast* pada (a) *rank* 0, (b) *rank* 1

4.3 Program *Gather* dan *Scatter*

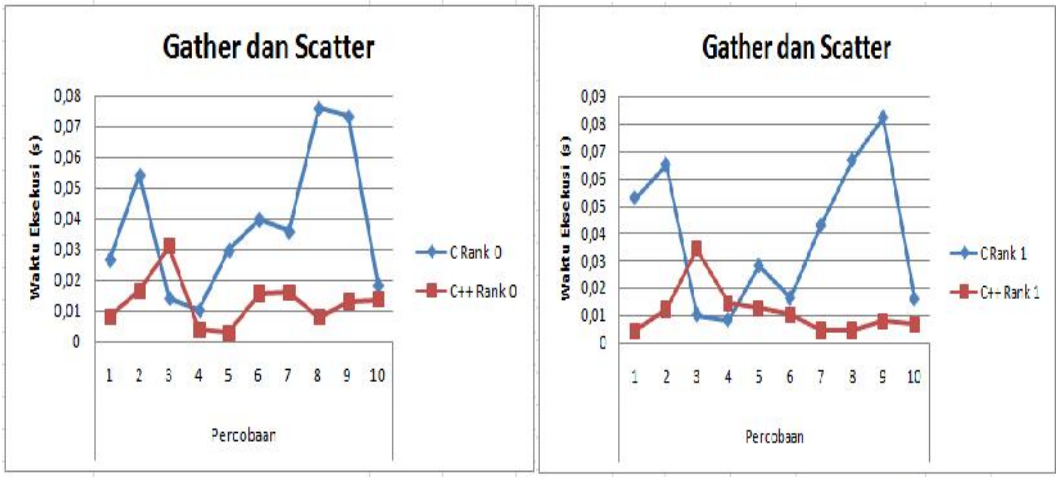
Berdasarkan Tabel 4.5 dan Tabel 4.6 diperoleh rata-rata waktu eksekusi program *Gather* dan *Scatter* yang telah di-*compile* pada saat *rank* 0 dikumpulkan dan *rank* 1 disebarkan, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Dari Tabel 4.5 dapat dibuat grafik perbandingan waktu eksekusi program *Gather* dan *Scatter* antara bahasa C dan bahasa C++ pada saat *rank* 0 dikumpulkan seperti terlihat pada Gambar 4.3. (a). Dan dari Tabel 4.6 dapat dibuat grafik perbandingan waktu eksekusi program *Gather* dan *Scatter* antara bahasa C dan bahasa C++ pada saat *rank* 1 disebarkan seperti terlihat pada Gambar 4.3. (b).

Tabel 4.5 Waktu eksekusi program *Gather* dan *Scatter* pada *rank* 0

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,027	0,054	0,014	0,01	0,029	0,039	0,036	0,076	0,07	0,018	0,0373
C++	0,008	0,016	0,03	0,004	0,003	0,015	0,016	0,008	0,013	0,014	0,0127
Selisih											0,0246

Tabel 4.6 Waktu eksekusi program *Gather* dan *Scatter* pada *rank* 1

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,053	0,065	0,01	0,008	0,03	0,016	0,043	0,07	0,08	0,016	0,0391
C++	0,004	0,012	0,034	0,014	0,013	0,01	0,005	0,005	0,008	0,007	0,0112
Selisih											0,0279



(a) (b)

Gambar 4.3 Grafik hasil eksekusi program *Gather* dan *Scatter* pada (a) *rank* 0, (b) *rank* 1

4.4 Program *AllGather*

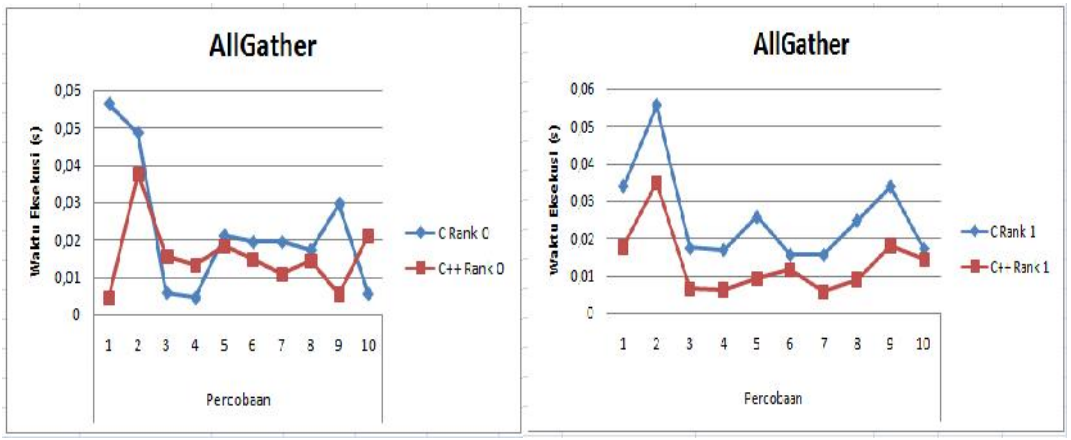
Berdasarkan Tabel 4.7 dan Tabel 4.8 diperoleh rata-rata waktu eksekusi program *AllGather* yang telah di-*compile* pada saat *rank* 0 dan *rank* 1 dikumpulkan, waktu eksekusi bahasa C lebih cepat daripada bahasa C++ seperti terlihat pada. Dari Tabel 4.7 dapat dibuat grafik perbandingan waktu eksekusi program *AllGather* antara bahasa C dan bahasa C++ pada saat *rank* 0 dikumpulkan seperti terlihat pada Gambar 4.4. (a). Dan dari Tabel 4.8 dapat dibuat grafik perbandingan waktu eksekusi program *AllGather* antara bahasa C dan bahasa C++ pada saat *rank* 1 dikumpulkan seperti terlihat pada Gambar 4.4. (b).

Tabel 4.7 Waktu eksekusi program *AllGather* pada *rank* 0

Bahasa	Percobaan										Rata-rata
	1	2	3	4	5	6	7	8	9	10	waktu (s)
C	0,056	0,05	0,006	0,004	0,021	0,019	0,019	0,017	0,03	0,006	0,0228
C++	0,004	0,038	0,015	0,013	0,018	0,014	0,01	0,014	0,005	0,021	0,0152
Selisih											0,0076

Tabel 4.8 Waktu eksekusi program *AllGather* pada *rank* 1

Bahasa	Percobaan										Rata-rata
	1	2	3	4	5	6	7	8	9	10	waktu (s)
C	0,034	0,055	0,017	0,017	0,025	0,015	0,015	0,025	0,034	0,017	0,0254
C++	0,017	0,035	0,006	0,006	0,009	0,011	0,005	0,009	0,018	0,014	0,013
Selisih											0,0124



(a) (b)

Gambar 4.4 Grafik hasil eksekusi program *AllGather* pada (a) *rank* 0, (b) *rank* 1

4.5 Program *AlltoAll*

Berdasarkan Tabel 4.9 diperoleh rata-rata waktu eksekusi program *AlltoAll* yang telah di-*compile* pada saat *rank* 0 mengirim data ke *rank* 1, waktu eksekusi bahasa C++ lebih cepat daripada bahasa C. Dari Tabel 4.9 dapat dibuat grafik perbandingan waktu eksekusi program *AlltoAll* antara bahasa C dan bahasa C++ pada saat *rank* 0 dikumpulkan seperti terlihat pada Gambar 4.5. (a).

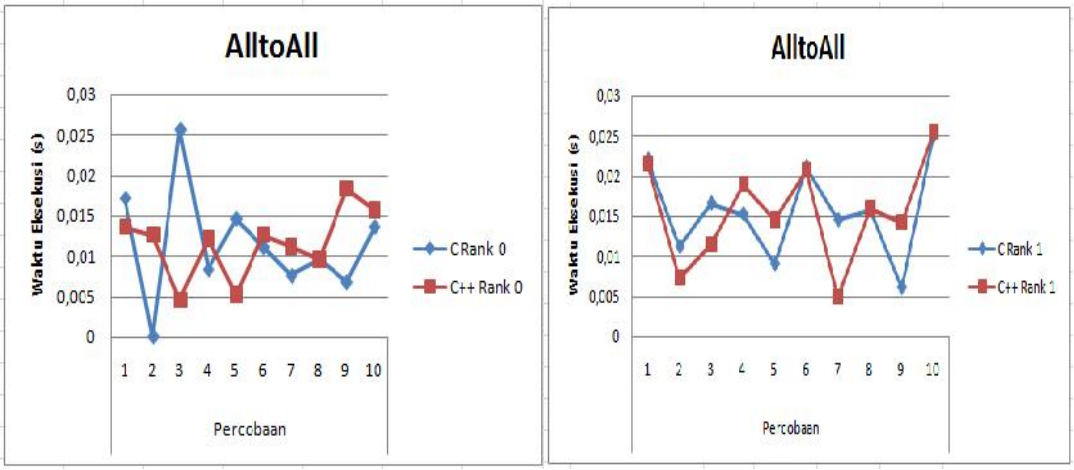
Tabel 4.9 Waktu eksekusi program *AlltoAll* pada *rank* 0

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,017	0,00008	0,026	0,008	0,014	0,01	0,007	0,009	0,007	0,014	0,011208
C++	0,013	0,013	0,005	0,012	0,005	0,013	0,01	0,009	0,018	0,016	0,0114
Selisih											-0,00019

Berdasarkan Tabel 4.10 diperoleh rata-rata waktu eksekusi program *AlltoAll* yang telah di-*compile* pada saat *rank* 1 menerima data dari *rank* 0, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Dari Tabel 4.10 dapat dibuat grafik perbandingan waktu eksekusi program *AlltoAll* antara bahasa C dan bahasa C++ pada saat *rank* 1 menerima data dari *rank* 0 seperti terlihat pada Gambar 4.5. (b).

Tabel 4.10 Waktu eksekusi program *AlltoAll* pada *rank* 1

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,02	0,01	0,016	0,015	0,009	0,021	0,014	0,015	0,006	0,025	0,0151
C++	0,02	0,007	0,01	0,018	0,014	0,02	0,005	0,016	0,014	0,025	0,0149
Selisih											0,0002



(a) (b)

Gambar 4.5 Grafik hasil eksekusi program *AlltoAll* pada (a) *rank* 0, (b) *rank* 1

4.6 Program *Reduce*

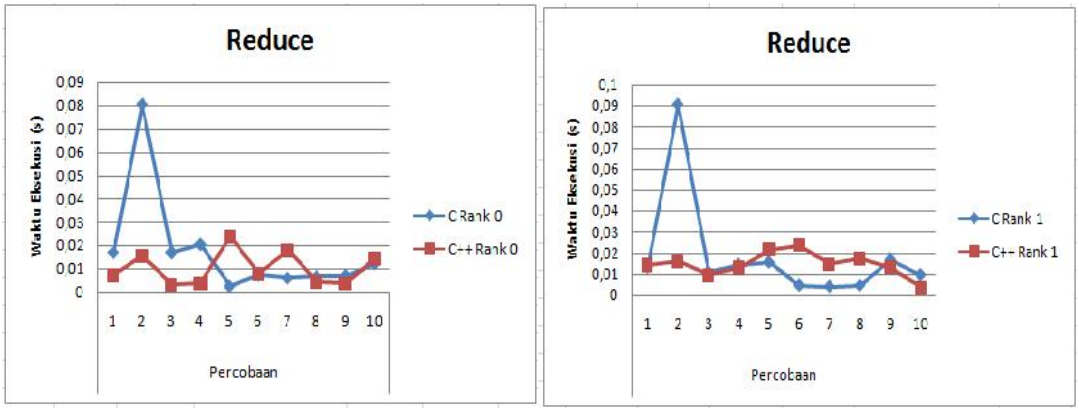
Berdasarkan Tabel 4.11 dan Tabel 4.12 diperoleh rata-rata waktu eksekusi program *Reduce* yang telah di-*compile* pada saat *rank* 0 dan *rank* 1 dikurangi, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Dari Tabel 4.11 dapat dibuat grafik perbandingan waktu eksekusi program *Reduce* antara bahasa C dan bahasa C++ pada saat *rank* 0 dikurangi seperti terlihat pada Gambar 4.6. (a). Dan dari Tabel 4.12 dapat dibuat grafik perbandingan waktu eksekusi program *Reduce* antara bahasa C dan bahasa C++ pada saat *rank* 1 dikurangi seperti terlihat pada Gambar 4.6. (b).

Tabel 4.11 Waktu eksekusi program *Reduce* pada *rank* 0

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,017	0,08	0,017	0,02	0,002	0,008	0,006	0,007	0,007	0,012	0,0176
C++	0,007	0,015	0,003	0,003	0,023	0,008	0,018	0,004	0,004	0,014	0,0099
Selisih											0,0077

Tabel 4.12 Waktu eksekusi program *Reduce* pada *rank* 1

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,013	0,09	0,01	0,014	0,015	0,004	0,004	0,004	0,017	0,009	0,018
C++	0,014	0,016	0,009	0,013	0,021	0,023	0,015	0,017	0,013	0,003	0,0144
Selisih											0,0036



(a) (b)

Gambar 4.6 Grafik hasil eksekusi program *Reduce* pada (a) *rank* 0, (b) *rank* 1

4.7 Program *Reduce Scatter*

Berdasarkan Tabel 4.13 diperoleh rata-rata waktu eksekusi program *Reduce Scatter* yang telah di-*compile* pada saat *rank* 0 digabungkan, waktu eksekusi bahasa C++ lebih cepat daripada bahasa C. Dari Tabel 4.13 dapat dibuat grafik perbandingan waktu eksekusi program *Reduce Scatter* antara bahasa C dan bahasa C++ pada saat *rank* 0 digabungkan seperti terlihat pada Gambar 4.7. (a).

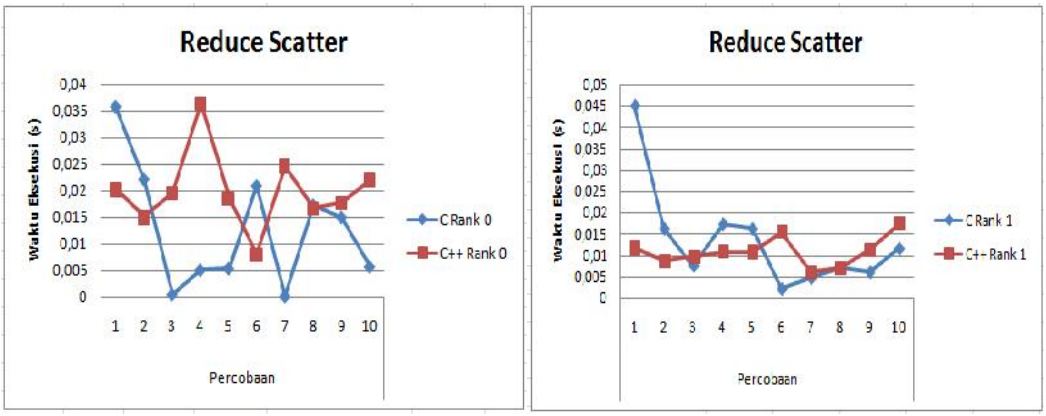
Tabel 4.13 Waktu eksekusi program *Reduce Scatter* pada *rank* 0

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,036	0,02	0,0005	0,005	0,005	0,02	0,0001	0,017	0,015	0,006	0,01246
C++	0,02	0,015	0,02	0,04	0,02	0,008	0,024	0,017	0,018	0,02	0,0202
Selisih											-0,00774

Berdasarkan Tabel 4.14 diperoleh rata-rata waktu eksekusi program *Reduce Scatter* yang telah di-*compile* pada saat *rank* 1 disebarkan, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Dari Tabel 4.14 dapat dibuat grafik perbandingan waktu eksekusi program *Reduce Scatter* antara bahasa C dan bahasa C++ pada saat *rank* 1 disebarkan seperti terlihat pada Gambar 4.7. (b).

Tabel 4.14 Waktu eksekusi program *Reduce Scatter* pada *rank* 1

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,045	0,02	0,007	0,017	0,016	0,002	0,005	0,007	0,006	0,012	0,0137
C++	0,012	0,008	0,01	0,01	0,01	0,015	0,006	0,007	0,01	0,017	0,0105
Selisih											0,0032



(a) (b)

Gambar 4.7 Grafik hasil eksekusi program *Reduce Scatter* pada (a) *rank* 0, (b) *rank* 1

4.8 Program *Scan*

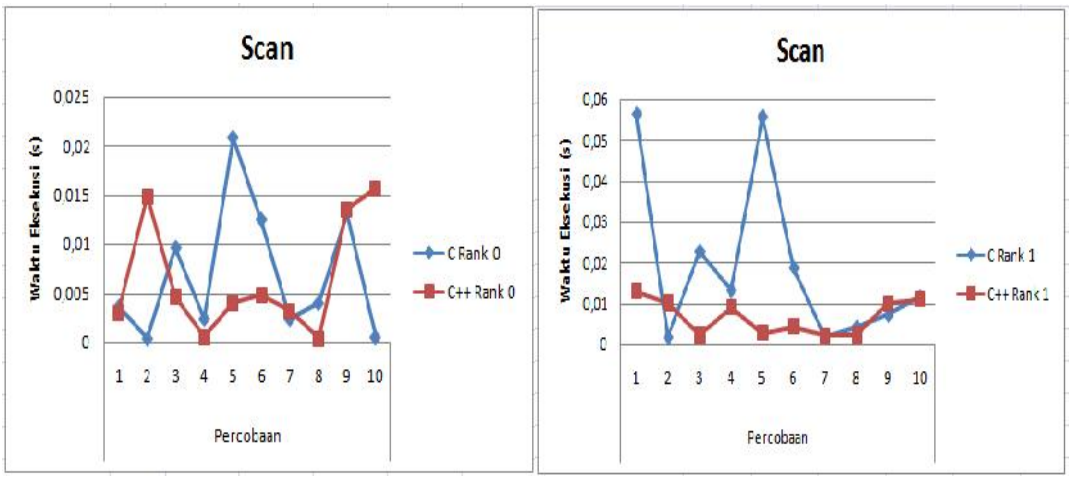
Berdasarkan Tabel 4.15 dan Tabel 4.16 diperoleh rata-rata waktu eksekusi program *Scan* yang telah di-*compile* pada saat *rank* 0 dan *rank* 1 dipindai, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Dari Tabel 4.15 dapat dibuat grafik perbandingan waktu eksekusi program *Scan* antara bahasa C dan bahasa C++ pada saat *rank* 0 dipindai seperti terlihat pada Gambar 4.8. (a). Dan dari Tabel 4.16 dapat dibuat grafik perbandingan waktu eksekusi program *Scan* antara bahasa C dan bahasa C++ pada saat *rank* 1 dipindai seperti terlihat pada Gambar 4.8. (b).

Tabel 4.15 Waktu eksekusi program *Scan* pada *rank* 0

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,004	0,0003	0,01	0,002	0,02	0,012	0,002	0,004	0,01	0,0005	0,00678
C++	0,003	0,015	0,005	0,0006	0,004	0,005	0,003	0,0004	0,01	0,016	0,0065
Selisih											0,00028

Tabel 4.16 Waktu eksekusi program *Scan* pada *rank* 1

Bahasa	Percobaan										Rata-rata waktu (s)
	1	2	3	4	5	6	7	8	9	10	
C	0,06	0,002	0,02	0,013	0,05	0,019	0,002	0,004	0,007	0,01	0,0187
C++	0,013	0,01	0,002	0,009	0,003	0,004	0,002	0,002	0,01	0,007	0,0062
Selisih											0,0125



(a) (b)

Gambar 4.8 Grafik hasil eksekusi program *Scan* pada (a) *rank* 0, (b) *rank* 1

BAB V
KESIMPULAN DAN SARAN

5.1 KESIMPULAN

Berdasarkan pengujian dan analisis dari penelitian ini, telah berhasil dibangun 1 *head node* dan 2 *compute node* yang dapat menjalankan kompilasi kode program paralel. Selain itu, diperoleh beberapa kesimpulan antara lain:

1. Pada kode program Sinkronisasi *Barrier*, waktu eksekusi saat *rank* 0 dan rank 1 ditahan, bahasa C++ lebih cepat daripada bahasa C dengan selisih waktu masing-masing 0,0011 s dan 0,0005 s.
2. Pada kode program *Broadcast*, waktu eksekusi saat *value* pada rank 0 dan rank 1 disiarkan, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu masing-masing 0,00142 s dan 0,0016 s.
3. Pada kode program *Gather* dan *Scatter*, waktu eksekusi saat data pada *rank* 0 dikumpulkan dan *rank* 1 disebarkan, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu masing-masing 0,0246 s dan 0,0279 s.
4. Pada kode program *AllGather*, waktu eksekusi saat *rank* 0 dikumpulkan, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu masing-masing 0,0076 s dan 0,0124 s.
5. Pada kode program *AlltoAll*, waktu eksekusi saat data dikirim dari *rank* 0 ke *rank* 1, bahasa C++ lebih cepat daripada bahasa C dengan selisih waktu 0,00019 s. Namun pada saat *rank* 1 menerima data dari *rank* 0, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu 0,0002 s.
6. Pada kode program *Reduce*, waktu eksekusi saat *value* pada rank 0 dan rank 1 dikurangi, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu masing-masing 0,0077 s dan 0,0036 s.
7. Pada kode program *Reduce Scatter*, waktu eksekusi saat nilai pada *rank* 0 digabungkan, bahasa C++ lebih cepat daripada bahasa C dengan selisih waktu 0,00774 s. Namun pada saat nilai pada *rank* 1 disebarkan, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu 0,0032 s.
8. Pada kode program *Scan*, waktu eksekusi saat data pada *rank* 0 dipindai, bahasa C lebih cepat daripada bahasa C++ dengan selisih waktu masing-masing 0,00028 s dan 0,0125 s.

9. Berdasarkan selisih waktu yang diperoleh dari program-program paralel yang dieksekusi, waktu eksekusi bahasa C lebih cepat daripada bahasa C++. Jadi fungsi - fungsi MPI dalam bahasa C++ tidak diperlukan lagi pada rutin komunikasi kolektif.

5.2 SARAN

Untuk penelitian dan melakukan pengembangan lebih lanjut, ada beberapa hal yang dapat dilakukan antara lain:

1. Penelitian lanjutan yang membandingkan waktu eksekusi dalam pemrograman paralel pada rutin tipe data turunan.
2. Disarankan mengukur *performance* berdasarkan *speed up* pada rutin komunikasi kolektif.